AFIT/ENC/GCS/93D-1

AIR POLLUTION TRANSPORT MODELING

THESIS
David Michael Paal
Captain, USAF

AFIT/ENC/GCS/93D-1

Approved for public release; distribution unlimited

AFIT/ENC/GCS/93D-1

AIR POLLUTION TRANSPORT MODELING

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science (Computer Systems)

David Michael Paal, B.A.
Captain, USAF

December 1993

Acknowledgements

I would like to thank the following people for their help in making this thesis effort a success: First, special thanks to Dr. Dennis Quinn, my thesis advisor, for his guidance, patience, and support over the last year and a half. His knowledge and guidance were immeasurably helpful in this thesis effort. I would also like to thank Major David Coulliette, a committee member, for his help and advice in the area of numerical methods. Discussions held with him on various other subjects were also very encouraging. My thanks to Dr. Henry Potoczny, a committee member, for his reviews of the thesis.

Finally, and most importantly, my most sincere and heartfelt thanks to Teresa Lesiak, my fiancée, for her love, and support during the last year. Her encouragement has given me the confidence to pursue this degree with all my effort.

David Michael Paal

Table of Contents

List of Figures

List of Tables

Abstract

This research effort addresses modeling of the transportation of air pollution in the atmosphere and the numerical analysis of the partial differential equations used in such modeling. Three Gaussian models are examined and compared using example problems. Several finite difference schemes are developed to solve the partial differential equations used in air pollution transport modeling. This study examines three Gaussian models: SCREEN, AFTOX, and the program GAUSPLUM. The model GAUSPLUM is developed in this study and uses the Ada programming language and the analytic solution to the advection-diffusion equation. Numerical analysis of several of the partial differential equations (PDE) used in air pollution modeling is also examined. The equations are generally parabolic or hyperbolic PDE's. The following are examined in this research: the advection equation; the one-, two-, and three-dimensional advection-diffusion equations; and the two-dimensional steady-state equation.

AIR POLLUTION TRANSPORT MODELING

I. Introduction

1.1 Background

During the last decade there has been an increasing public interest and concern about environmental issues, in particular, air pollution. Zannetti (29) stated some of the issues of interest: 1) the "greenhouse" effect, which could cause an increase in the earth's average temperature due to increasing concentrations of carbon dioxide in the atmosphere; 2) the possible depletion of the ozone layer, a natural screen of harmful solar radiation, by certain pollutant species emitted by anthropogenic activities; 3) indoor air pollution, such as asbestos and radon gas; 4) nuclear disasters, especially after the Chernobyl disaster; 5) atmospheric visibility and its impairment by air pollutants; and 6) risk assessment, especially in the prevention of accidental releases of toxic pollutants. One of the simplest air pollution transport models is a plume model (more specifically, a Gaussian plume model). Many air pollution plume models have been developed to determine the concentrations of pollutants in the air and the overall quality of the atmosphere. These plume models are mostly "specific" in nature; that is they were developed with a specific pollutant, source of pollutant, or change of concentration in mind.

1.2 Problem

Plume models have been used extensively in determining and predicting changes in concentrations of air pollutants. Most models use analytical solutions to the partial differential equations used in pollution modeling in their calculations. This study

will analyze the following Gaussian plume models: SCREEN (10), AFTOX (19), and GAUSPLUM (the program developed in this research), all of which use analytical solutions in their calculations. This study will also calculate numerical solutions of the advection, steady state, and advection-diffusion equations using finite difference methods in an attempt to get solutions in cases of low wind speed which the Gaussian plume model does not handle.

1.3 Summary of Current Knowledge

Since the middle 1980s, researchers have addressed the cause and effect relationships derived from air pollution models in general (20). The results of this research has sparked interest in studying cause and effect relationships in plume modeling and specifically in urban areas. Much work has been done on plume rise and diffusion models, (14, 15, 21) and on modeling pollution in an urban environment (9, 8). There have also been studies conducted which evaluated and verified mathematical models, and these studies will be further discussed in Chapter II.

1.4 Scope

It is not the intent of this study to completely explain all plume models or all air pollution transport models. Such an explanation would be beyond the scope of this thesis. The thesis is limited to the study of some specific plume models. The scope of this research is limited to the following:

- The development of a basic pollutant transport model using the Ada programming language.
- A description of the mathematical aspects of air pollution transport models.
- An analysis and comparison of the numerical results of the models.
- Conclusion reached from the results, and any recommendations made for future research.

1.5 Approach

The basic approach taken in this research follows:

1. This study describes the general mathematical formula, the Gaussian plume equation, used in plume models.
2. The research examines the specific differences among the models and how the general model was modified as a result of such differences. Most of the differences are due to where the model is intended to be used, and how conservative the user wants to be. Location, terrain, climate, and type of source are also criteria specific to each model.
3. Example problems will then be applied to the models and a comparison of the results will be conducted.
4. The research will evaluate several finite difference schemes and apply them to solving the advection equation, the one-, two-, and three-dimensional advection-diffusion equations, and the two-dimensional steady state equation.

1.6 Materials and Equipment

The SCREEN model used in this research was obtained from the United States Department of Commerce, National Technical Information Service, Computer Products Division in Springfield, Virginia via the United States Environmental Protection Agency's Support Center for Regulatory Air Models Bulletin Board System (SCRAM-BBS) with the Bulletin Board number (919) 541-5742. AFTOX was obtained from faculty in the physics department at the Air Force Institute of Technology (AFIT).

The computer facilities at AFIT are sufficient to run the software models used in this research. No other computer equipment or support is needed.

1.7 Summary of Thesis

The organization of this thesis follows:

Chapter II gives an overview of current literature concerning plume models, and numerical solutions to advection and diffusion equations.

Chapter III discusses the overall methodology of this research in plume modeling including the numerical methods used to find solutions to the advection and diffusion equations.

Chapter IV shows the results of the research and discusses the comparisons made.

Chapter V discusses the conclusions derived from this research and recommendations for future research.

Appendix A contains the Ada source code for the basic plume model, GAUSPLUM, developed in this research.

Appendix B contains the Ada source code for the advection and advection-diffusion equations.

Appendix C describes the one-dimensional diffusion equation, and the three-dimensional steady state equation.

II. Literature Review

2.1 Introduction

This literature review examines current research in the areas of pollution modeling and methods of finding numerical solutions of the advection and diffusion equations. Equation (2.1) shows the three dimensional advection-diffusion equation on which most models are based.

$$\frac{\partial c}{\partial t} + \bar{u}\frac{\partial c}{\partial x} = k_x\frac{\partial^2 c}{\partial x^2} + k_y\frac{\partial^2 c}{\partial y^2} + k_z\frac{\partial^2 c}{\partial z^2} \tag{2.1}$$

Various approaches to air pollution plume modeling are described in the literature (14, 7). This review addresses the Gaussian and Lagrangian plume models. The Gaussian plume model is the most common air pollution model used because of its relative ease of use with easily measurable meteorological parameters (29). One use of the Gaussian plume equation is found in the trajectory plume model. Okamoto's research of trajectory plume modeling (22), and Schohl's research of Lagrangian plume modeling (24) is addressed in this literature review. A variety of numerical and approximate solutions to plume modeling are also described in the literature (13, 17, 16, 23). This literature review will also look at some numerical methods and some of the limits of pollution modeling.

Two terms that need further definition are stability condition and stability class:

A stability condition is a constraint that is put on variables in a solution method. When the value of the variables are in the range of the constraint then small changes in the variables produces small changes in the results and the method is said to be stable (3).

A stability class is a parameter used to represent the atmospheric conditions (ranging from extremely unstable to extremely stable, A to G). It is determined

using the standard deviation of the horizontal wind direction fluctuation and the wind speed at 10 meters in meters per second. See Turner's workbook (27) or Table 3 in the AFTOX User's Guide (19) for classifications.

Stability condition and stability class are common terms used in the analysis of air pollution transport modeling.

2.2 Trajectory plume model

Among the different models used for calculating pollutant concentrations, a Gaussian plume model is the most common. The Gaussian plume model is based on a formula that describes the three-dimensional pollutant concentration generated by a point source under stationary meteorological and emission conditions (22, 27). Its calculation error becomes a serious problem under weak, variable-direction wind conditions. It is for this reason that Okamoto's (22) research of a trajectory plume model or a plume segment model was conducted. This model treats time varying transport conditions and changes in wind direction and speeds. In the segmented plume approach, the plume is broken up into independent elements (plume segments or sections) whose initial features and time dynamics are a function of emission conditions and local meteorological conditions encountered by the plume segment (29:165-7). Segments are sections of a Gaussian plume. Therefore, the concentration of air pollution for each segment can be calculated using the Gaussian plume model. The trajectory plume model then adds segments together to get concentration distributions over the entire plume.

2.3 Lagrangian plume model

Another model which uses segmented plumes to calculate concentrations of air pollutants is the Lagrangian box model. The Lagrangian box model breaks the plume into boxes (segments), each of which follows the average wind flow. Because the boxes move with the average wind flow, the Lagrangian box model provides concentration

outputs along trajectories and consequently comparisons with concentrations at fixed grid points of traditional finite difference or finite element solutions are difficult (29). Schohl (24) discusses an interpolation method to use in a Lagrangian scheme:

> "Lagrangian schemes for numerical approximation of concentration amounts are capable of providing satisfactory accuracy with minimal computational effort. The overall accuracy of these schemes is dependent on the interpolation method used to obtain the approximation."(24)

Schohl's (24) research shows that the cubic-spline interpolation method is one which provides better than average accuracy to the Lagrangian model. For more information on cubic spline interpolation see references (3:126-7) and (28). This interpolation provides a fourth order approximation with minimal computational effort.

2.4 Numerical and approximate solutions

2.4.1 Plume rise model solution method. Krishnamurthy (18) states that little seems to have been published which gives useful guidelines concerning the accuracy of approximations used in air pollution transport models. In fact, he says

> "In general, the application of plume models requires numerical solutions of the governing conservation equations which are highly non-linear. Rather than obtain such "exact" solutions it is easier to use various approximations to them. Sometimes, however, the accuracy of such approximations relative to the exact solution is quite uncertain." (18:2083)

Krishnamurthy's research calculates numerical solutions using the plume rise model of Hoult, Fay, and Forney (15). Various asymptotic approximations are assessed over a wide range of parameters. Mass, momentum, energy, turbulence, temperature, and wind speed and direction are the parameters used in these approximations. Comparisons of the numerical and approximate solutions are made and shown to agree over a fairly wide range of the parameters.

2.4.2 Advection equation solution methods. Many methods are described in the literature which numerically solve partial differential equations. In partic-

ular, Chock, in "A Comparison of Numerical Methods for Solving the Advection Equation–I, II, and III"(4, 5, 6) compares several algorithms and variations of each which are used for solving the two-dimensional advection equation which is

$$\frac{\partial c}{\partial t} + u\frac{\partial c}{\partial x} + v\frac{\partial c}{\partial y} = 0 \tag{2.2}$$

where c is the concentration, and u and v are the x and y components of the wind speed.

The methods are compared in terms of accuracy, speed, and storage requirements. Recommendations are made in the articles as to how useful the methods would be for air quality modeling.

2.4.3 Diffusion equation solution methods. There are also many techniques in the literature for solving the atmospheric diffusion equation

$$\frac{\partial c}{\partial t} = k_x\frac{\partial^2 c}{\partial x^2} + k_y\frac{\partial^2 c}{\partial y^2} + k_z\frac{\partial^2 c}{\partial z^2} \tag{2.3}$$

where in equation (2.3) c is the concentration of the pollutant in question and k_x, k_y, and k_z are the x, y, and z diffusivity terms respectively. McRae, Goodin, and Seinfeld look at several techniques for solving both the advection equation (2.2) and the diffusion equation (2.3) in their one-dimensional form in "Numerical Solution of the Atmospheric Diffusion Equation for Chemically Reacting Flows" (21).

2.4.4 Advection-Diffusion solution methods. The one-dimensional advection-diffusion equation, or as Strikwerda (26:129-31) calls it, the convection- diffusion equation, is

$$\frac{\partial c}{\partial t} + \bar{u}\frac{\partial c}{\partial x} = k_x\frac{\partial^2 c}{\partial x^2} \tag{2.4}$$

where c is the concentration, $\bar{u}$ is the advection coefficient and k_x is the diffusivity coefficient. Strikwerda looks at this equation two ways; one way using a substitution of $y = x - \bar{u}t$ with $w(t, y) = c(t, y + \bar{u}t)$, and the other way using the forward-time central space finite difference method (see Section 3.4.2 for this method).

2.5 Limits of air pollution modeling

Numerical and approximate solutions may agree over a wide range of parameters, but the limits of meteorological modeling can lead to uncertainty in atmospheric parameters such as turbulence, wind and temperature. This can lead to uncertainty in the models that use these parameters. The Gaussian plume model doesn't perform well when the wind is weak, and an uncertainty in the wind parameter can add to this poor performance. Benarie (1) states that "turbulence is a meteorological quantity that can only be approximated and then only in the most ideal circumstances". Meteorological conditions are inherently variable and this variability causes an uncertainty in the accuracy of meteorological parameters. Plume models which incorporate these parameters inherit this uncertainty. Therefore, numerical and approximate solutions of air pollution models are dependent on the limits of meteorological modeling used to calculate the parameters for pollution models.

2.6 Conclusion

Trajectory plume modeling and Lagrangian plume modeling both use segmentation of pollution plumes in their calculation of pollutant concentrations. The trajectory model uses the Gaussian plume equation to calculate the concentration in each segment. The Lagrangian model uses interpolation methods for calculations between segments. Many methods for solving advection, diffusion, and advection-diffusion equations used in these models are found in the literature including, the plume rise model, several advection equation solution methods in Chock's research, techniques to solve both one-dimensional advection and diffusion equations

in McRae's research, and methods used by Strikwerda to solve the one-dimensional advection-diffusion equation. However, the accuracy of these numerical and approximate solutions are dependent on the parameters used in the models. Much more research is needed in the areas of numerical and approximate solutions and the determination of the parameters used in pollution models. This thesis will use finite difference methods to solve these equations.

III. Methodology

3.1 Introduction

The purpose of this study is to examine air pollution transport models and the partial differential equations used in air pollution modeling. This research examines the following models: SCREEN (10), AFTOX (19), the program GAUSPLUM developed in this study, and numerical schemes used to solve the advection, diffusion, two-dimensional steady state, and one, two and three dimensional advection-diffusion equations.

The programming language, a brief description of the capabilities, and the mathematical equations used in SCREEN, AFTOX, and GAUSPLUM are discussed. The two primary capabilities used in this research include finding the concentration of a pollutant at a given location, and finding the location and value of the maximum concentration. A comparison of these capabilities in each model is done using six example problems taken from the Workbook of Atmospheric Dispersion Estimates (27). These problems are described in detail below.

The Lax-Friedrichs and leapfrog finite difference methods will be used on the one-dimensional advection equation. The forward-time central-space finite difference method will be used on the advection-diffusion equations.

3.2 Description of Gaussian plume models

This section describes the SCREEN and AFTOX models, and the GAUSPLUM model developed in this research.

3.2.1 The SCREEN model. SCREEN is written in FORTRAN programming language. SCREEN estimates pollutant concentration from continuous sources using a Gaussian plume model that incorporates source-related factors, such as emission rate, stack gas temperature, stack height, stack inside diameter, and stack gas

exit velocity, and meteorological factors, such as ambient temperature, wind speed and direction. The Gaussian model equations are described in Turner's workbook (27).

The basic equation, for determining ground-level concentrations under the centerline of the plume, used in the SCREEN model (10) is:

$$
\begin{aligned}
c = [q/(2\pi u_s \sigma_y \sigma_z)] \quad & \cdot(\exp[-\tfrac{1}{2}((z_r - h_e)/\sigma_z)^2] \\
& + \exp[-\tfrac{1}{2}((z_r + h_e)/\sigma_z)^2] \\
& + \textstyle\sum_{N=1}^{k} [\ \exp[-\tfrac{1}{2}((z_r - h_e - 2Nz_i)/\sigma_z)^2] \\
& \qquad + \exp[-\tfrac{1}{2}((z_r + h_e - 2Nz_i)/\sigma_z)^2] \\
& \qquad + \exp[-\tfrac{1}{2}((z_r - h_e + 2Nz_i)/\sigma_z)^2] \\
& \qquad + \exp[-\tfrac{1}{2}((z_r + h_e + 2Nz_i)/\sigma_z)^2]])
\end{aligned}
\tag{3.1}
$$

where

c = concentration (g/m^3)
q = emission rate (g/s)
$\pi = 3.14159$
u_s = stack height wind speed (m/s)
σ_y = lateral dispersion parameter (m)
σ_z = vertical dispersion parameter (m)
z_r = receptor height above ground (m)
h_e = plume centerline height (m)
z_i = mixing height (m)
k = summation limit for multiple reflections of plume off of the ground and elevated inversion, usually ≤ 4

This equation accounts for the multiple eddy reflections from both the ground and the stable layer (z_i) and was suggested by Bierly and Hewson (2). The derivation of equation (3.1) is discussed in Section 3.2.3.

3.2.2 The AFTOX model. AFTOX is a program written in the Basic programming language. The two parts of the USAF Toxic Chemical Dispersion Model

(AFTOX) used in this research are the calculation of the toxic chemical concentration at a specific location and the strength and location of the maximum concentration. Refer to AFGL-TR-88-0009 (19) for further uses of the model. AFTOX uses the Gaussian puff equation, and the Gaussian plume equation in its calculations. These equations assume a Gaussian distribution of concentration and conservation of the pollutant during transport and diffusion.

The AFTOX model uses three basic models, the Gaussian puff equation (see eq.3.2), the Gaussian puff equation when an inversion is present (see eq.3.3), and the Gaussian plume model (see eq.3.4).

The Gaussian puff model is

$$\begin{aligned} c(x,y,z,t-\tilde{t}) = \; & \left[q(\tilde{t})/((2\pi)^{3/2}\sigma_x\sigma_y\sigma_z)\right] \\ & \cdot \exp[-\tfrac{1}{2}((x-u(t-\tilde{t}\,))/\sigma_x)^2] \\ & \cdot \exp[-(y/\sigma_y)^2/2] \\ & \cdot (\exp[-\tfrac{1}{2}((z-H)/\sigma_z)^2] + \exp[-\tfrac{1}{2}((z+H)/\sigma_z)^2]) \end{aligned} \tag{3.2}$$

where

c is concentration in the puff at (x,y,z) at $(t-\tilde{t})$
q is total mass of the puff
u is wind speed at 10m
σ_x is downwind dispersion parameter
σ_y is lateral dispersion parameter
σ_z is vertical dispersion parameter
t is total elapsed time of pollution emission
$\tilde{t}$ is time of puff emission
$(t-\tilde{t})$ is elapsed time since puff emission
H is height of the source.

The Gaussian puff model with an inversion has the addition of the following expression to the last two terms of equation (3.2)

$$\begin{aligned}\sum_N(\quad & \exp[-\tfrac{1}{2}((z-H-2NL)/\sigma_z)^2] + \exp[-\tfrac{1}{2}((z+H-2NL)/\sigma_z)^2] + \\ & \exp[-\tfrac{1}{2}((z-H+2NL)/\sigma_z)^2] + \exp[-\tfrac{1}{2}((z+H+2NL)/\sigma_z)^2]).\end{aligned} \tag{3.3}$$

where L is the mixing layer height and N is the number of reflections caused by the inversion. This equation is similar to equation (3.1) used in the SCREEN model.

The Gaussian plume model used in AFTOX is

$$c = \frac{q}{2\pi\sigma_x\sigma_y u} \cdot \exp[-\frac{1}{2}(\frac{y}{\sigma_y})^2] \cdot (\exp[-\frac{1}{2}(\frac{z-H}{\sigma_z})^2] + \exp[-\frac{1}{2}(\frac{z+H}{\sigma_z})^2]). \tag{3.4}$$

The derivation of equations (3.2), (3.3), and (3.4) is discussed in Section 3.2.3.

3.2.3 The GAUSPLUM model from this research. GAUSPLUM is written in the Ada programming language. It uses the Gaussian plume equation with the Pasquill-Gifford σ_y and σ_z described by Zannetti (29:149-50).

The basic Gaussian plume equation used in GAUSPLUM is:

$$c = \frac{q}{2\pi\sigma_y\sigma_z\bar{u}} \exp[-\frac{1}{2}(\frac{y_r}{\sigma_y})^2] \exp[-\frac{1}{2}(\frac{h_e - z_r}{\sigma_z})^2] \tag{3.5}$$

where c is the concentration at $r = (x_r, y_r, z_r)$ due to emissions from the source at (x_s, y_s, z_s); q is the emission rate; σ_y and σ_z are the horizontal and vertical, respectively, dispersion parameters; $\bar{u}$ is the horizontal wind speed; and h_e is the effective emission height. The coordinate system in this model has the x-axis in the $\bar{u}$ direction.

Equations (3.1), (3.2), (3.3), (3.4), and (3.5) can be derived from the three-dimensional advection-diffusion equation which is

$$c_t + \bar{u}c_x = k_x c_{xx} + k_y c_{yy} + k_z c_{zz}. \tag{3.6}$$

In equation (3.6), at steady state, $c_t = 0$ so the equation reduces to

$$\bar{u}c_x = k_x c_{xx} + k_y c_{yy} + k_z c_{zz}. \tag{3.7}$$

For many air pollution transport problems, the $k_x c_{xx}$ term is negligable compared to the $\bar{u}c_x$ (25:542-543). Then equation (3.7) reduces to

$$\bar{u}c_x = k_y c_{yy} + k_z c_{zz}. \tag{3.8}$$

If $\bar{u}$, k_y, and k_z are constant and the source is a point source, an exact closed form solution of equation (3.8) can be obtained using Fourier transform techniques (25:556). This solution is

$$c(x,y,z) = \frac{q}{4\pi(k_y k_z)^{1/2} x} \exp[-\frac{\bar{u}}{4x}(\frac{y^2}{k_y} + \frac{z^2}{k_z})]. \tag{3.9}$$

See Seinfeld page 543 and 556 for details. If $(\sigma_y^2 \bar{u})/(2x)$ and $(\sigma_y^2 \bar{u})/(2x)$ are both constant, then letting $k_y = (\sigma_y^2 \bar{u})/(2x)$ $k_z = (\sigma_z^2 \bar{u})/(2x)$ and substituting into equation (3.9) gives

$$c = \frac{q}{2\pi\sigma_y\sigma_z\bar{u}} \exp[-\frac{1}{2}(\frac{y}{\sigma_y})^2 - \frac{1}{2}(\frac{z}{\sigma_z})^2]. \tag{3.10}$$

Equation (3.10) is the same as equations (3.1), (3.2), (3.3), (3.4), and (3.5) except they include a term, H or h_e, in the z term to account for the height of the source. Equation (3.1) also includes a term for when an inversion is present in the atmosphere and assumes $y = 0$ so there is no y term in the equation. It is these equations which can be derived from the three-dimensional advection-diffusion equation that are used in the SCREEN, AFTOX, and GAUSPLUM models to solve the following problems.

3.3 Description of example problems

This section describes the problems taken from Turners workbook used to compare the three models.

- Problem 1 from the workbook shows a ground-level calculation directly downwind at a distance of 3000 meters. The ground-level source emits 3 g/sec of oxides of nitrogen with no effective rise. It is an overcast night with a 7 m/sec wind speed. This indicates a stability class D.
- Problem 2 from the workbook calculates a concentration 500 meters directly downwind from a source with an effective height of 60 meters. It is an overcast winter morning at 0800. The source emits an estimated 80 g/sec of sulfur dioxide into a wind of 6 m/sec. The stability class is still D.
- Problem 3 has the same conditions as problem 2 at the same distance downwind but at a distance of 50 meters from the x-axis. The SCREEN program only calculates concentrations directly downwind from the source so it does not apply to this problem.
- Problem 4 has a source emitting 151 g/sec at an effective height of 150 meters. This problem asks for the distance and the value of the maximum ground-level concentration on a sunny summer afternoon with a 10 meter wind speed of 4 m/sec from the northeast. This is a class-B stability.
- Problem 5 has the same conditions as problem 4 except it is on an overcast day. The stability class becomes D for this problem.
- Problem 11 in the workbook has the same conditions as problem 4 except it asks for the distance and value of the maximum concentration on the plume centerline on a clear night with a wind speed of 4 m/sec. This give stability class E.

These problems will be used to compare SCREEN, AFTOX, and GAUSPLUM while finite difference schemes will be used to solve the following mathematical models and compare the solutions to exact analytical solutions.

3.4 Mathematical models

This section describes the mathematical finite difference schemes examined in this research. The advection equation, the one, two and three dimensional advection-diffusion equations, and the two and three dimensional steady-state equations are described here.

3.4.1 The advection equation: The advection equation (see eq. 3.11), also known as the one-way wave equation, is a hyperbolic partial differential equation. This subsection describes the simple advection equation and the numerical methods used to solve it in this research. The advection equation is

$$\frac{\partial c}{\partial t} + \bar{u}\frac{\partial c}{\partial x} = 0 \tag{3.11}$$

also written as $c_t + \bar{u}c_x = 0$, where the subscript denotes differentiation, i.e., $c_t = \partial c/\partial t$, and $\bar{u}$ is the average wind speed.

Two finite difference schemes are used on the following initial boundary value problem:

$$c_t + c_x = 0 \quad \text{on} -2 \leq x \leq 3, 0 \leq t \tag{3.12}$$

with initial data

$$c_0(x) = \begin{cases} 1 - |x| & \text{if } |x| \leq 1 \\ 0 & \text{if } |x| \geq 1 \end{cases} \tag{3.13}$$

The boundary condition at time t is

$$c(x,t) = 0 \tag{3.14}$$

when $x = -2$, or $x = 3$.

The finite difference schemes to be used on the above initial-boundary value problem are the Lax-Friedrichs scheme (3.15) and the leapfrog scheme (3.16) as found in Strikwerda (26:13-4). The notation c_m^n is the same as $c(t_n, x_m)$ or in other words it is the value of c at the grid point (t_n, x_m).

$$\frac{c_m^{n+1} - \frac{1}{2}(c_{m+1}^n + c_{m-1}^n)}{\Delta t} + \bar{u}\frac{c_{m+1}^n - c_{m-1}^n}{2\Delta x} = 0 \tag{3.15}$$

$$\frac{c_m^{n+1} - c_m^{n-1}}{2\Delta t} + \bar{u}\frac{c_{m+1}^n - c_{m-1}^n}{2\Delta x} = 0 \tag{3.16}$$

where $\bar{u} = 1$ for this problem, Δt is the time incrementer, Δx is the space incrementer, m and n are integer grid counters for x and t respectively. Solving the schemes for c_m^{n+1} gives a linear combination of c at levels n and $n-1$. The Lax-Friedrichs scheme (3.15) can be written as

$$c_m^{n+1} = \frac{1}{2}(c_{m+1}^n + c_{m-1}^n) - \frac{1}{2}\bar{u}\lambda(c_{m+1}^n - c_{m-1}^n) \tag{3.17}$$

where $\lambda = \Delta t/\Delta x$. Likewise, the leapfrog scheme (3.16) can be written as

$$c_m^{n+1} = c_m^{n-1} - \bar{u}\lambda(c_{m+1}^n - c_{m-1}^n) \tag{3.18}$$

again with $\lambda = \Delta t/\Delta x$, and $\bar{u} = 1$.

The stability condition for the Lax-Friedrichs method is $\bar{u}\lambda \leq 1$, or since $\bar{u} = 1$ in this case, $\lambda \leq 1$. It is found by replacing c_m^n with $g^n e^{im\theta}$ in equation(3.17) and solving for g which yields

$$g = (e^{i\theta} + e^{-i\theta})/2 \; - \bar{u}\lambda(e^{i\theta} - e^{-i\theta})/2 \tag{3.19}$$

where $i = (-1)^{1/2}$. The quantity g is called the amplification factor. This is equivalent to

$$g = \cos\theta - i\bar{u}\lambda \sin\theta. \tag{3.20}$$

The stability condition $|g| \leq 1$ comes from Theorem 2.2.1 in Strikwerda (26:42). Note that $|g|^2$ is

$$|g|^2 = \; \cos^2\theta + \bar{u}^2\lambda^2 \sin^2\theta. \tag{3.21}$$

Therefore, $|g| \leq 1$ if $|\bar{u}\lambda| \leq 1$. Thus the stability condition $\bar{u}\lambda \leq 1$ since both $\bar{u}$ and λ are positive in this case.

If $\bar{u} = 1$, and $\lambda = 0.8$ the stability condition is satisfied for both schemes. Consistency occurs when the local truncation error vanishes as $\Delta x \rightarrow 0$ and $\Delta t \rightarrow 0$ (28:606). The local truncation error is the difference between the solution of the difference equation at a point, and the solution of the differential equation at the same point. By substituting Taylor series expansions into the Lax-Friedrichs and leapfrog schemes it can be shown that both schemes are consistent (26:21-23). Both conditions, stability and consistency, must be met for the scheme to be convergent. Both the Lax-Friedrichs scheme and the leapfrog scheme are, therefore, convergent for $\lambda = 0.8$ since both stability and consistency conditions are met. Figures (4.3) and (4.4) in Section 4.3.1 show that both schemes are convergent for $\lambda = 0.8$.

When $\lambda = 1.6$ is used for the Lax-Friedrichs method it is not convergent. This is because the stability condition is not met. However, as $\Delta x \rightarrow 0$ and $\Delta t \rightarrow 0$ the solution of the scheme does approach the solution of the advection equation, as long as $k^{-1}h^2 \rightarrow 0$, so the scheme is consistent (see Strikwerda, example 1.4.2, page

21.)(26:21). The results of using these finite difference schemes will be discussed in Section 4.3.1.

3.4.2 The 1-D advection-diffusion equation: The one-dimensional advection-diffusion equation (3.22) is also known as the one-dimensional convection-diffusion equation,

$$c_t + \bar{u}c_x = k_x c_{xx} \tag{3.22}$$

where k_x is a positive number and we assume that $\bar{u}$ is also positive. The forward-time central-space scheme used to solve equation (3.22) is

$$\frac{c_m^{n+1} - c_m^n}{\Delta t} + \bar{u}\frac{c_{m+1}^n - c_{m-1}^n}{2\Delta x} = k_x \frac{c_{m+1}^n - 2c_m^n + c_{m-1}^n}{\Delta x^2} \tag{3.23}$$

which is equivalent to

$$c_m^{n+1} = (1 - 2k_x\mu)c_m^n + k_x\mu(1-\alpha)c_{m+1}^n + k_x\mu(1+\alpha)c_{m-1}^n \tag{3.24}$$

where $\mu = \Delta t/\Delta x^2$ and $\alpha = \bar{u}\Delta x/2k_x$. The finite difference scheme is used on the following initial-boundary value problem:

$$c_t + \bar{u}c_x = k_x c_{xx} \quad \text{on} -2 \le x \le 3, 0 \le t \tag{3.25}$$

with initial data

$$c_0(x) = \begin{cases} 1 - |x| & \text{if } |x| \le 1 \\ 0 & \text{if } |x| \ge 1. \end{cases} \tag{3.26}$$

The boundary condition at time t is

$$c(x,t) = 0 \tag{3.27}$$

when $x = -2$, or $x = 3$.

The stability condition for this method is $k_x\mu \leq 1/2$. The stability analysis is similar to that done on the advection equation above, i.e. by replacing c_m^n with $g^n e^{im\theta}$ in equation (3.23) and solving for g gives

$$g = 1 - 4k_x\mu \sin^2 \frac{1}{2}\theta - i\bar{u}\lambda \sin\theta. \tag{3.28}$$

Using the condition $|g| \leq 1$ from Theorem 2.2.1 and Theorem 2.2.3 and Corollary 2.2.2 in Strikwerda (26), which shows that the first derivative term can be ignored, gives $4k_x\mu \sin^2 \frac{1}{2}\theta \leq 2$ and so $k_x\mu \leq 1/2$. Thus this method is conditionally stable. The results of using this finite difference scheme to solve equation (3.22) will be presented in Section 4.3.2.

3.4.3 The 2-D advection-diffusion equation: The two–dimensional advection-diffusion equation (see eq. 3.29) includes diffusion in both the downwind (x-axis) and crosswind (y-axis) directions. Advection is assumed to be negligable in the crosswind direction since the wind direction is in the x-axis direction. Therefore, the only advection term is $\bar{u}c_x$.

$$c_t + \bar{u}c_x = k_x c_{xx} + k_y c_{yy} \tag{3.29}$$

where $\bar{u}$ is the wind speed (positive constant), and $k_x = k_y$ are constant in this research and are the x-axis and y-axis diffusivity terms respectively. The forward-time central-space finite difference scheme used to solve equation (3.29) is

$$\begin{aligned} \frac{c_{m,p}^{n+1} - c_{m,p}^n}{\Delta t} + \bar{u}\frac{c_{m+1,p}^n - c_{m-1,p}^n}{2\Delta x} = \quad & (k_x/\Delta x^2)(c_{m+1,p}^n - 2c_{m,p}^n + c_{m-1,p}^n) \\ & + (k_y/\Delta y^2)(c_{m,p+1}^n - 2c_{m,p}^n + c_{m,p-1}^n) \end{aligned} \tag{3.30}$$

which is equivalent to

$$\begin{aligned} c_{m,p}^{n+1} = \quad & c_{m,p}^{n} - \tfrac{1}{2}\bar{u}\lambda(c_{m+1,p}^{n} - c_{m-1,p}^{n}) \\ & + k_x\alpha(c_{m+1,p}^{n} - 2c_{m,p}^{n} + c_{m-1,p}^{n}) \\ & + k_y\beta(c_{m,p+1}^{n} - 2c_{m,p}^{n} + c_{m,p-1}^{n}) \end{aligned} \tag{3.31}$$

where $c_{m,p}^{n} = c(x_m, y_p, t_n) = c(m\Delta x, p\Delta y, n\Delta t)$, $\lambda = \Delta t/\Delta x$, $\alpha = \Delta t/\Delta x^2$ and $\beta = \Delta t/\Delta y^2$.

The finite difference scheme is used on the following initial- boundary value problem:

$$c_t + \bar{u}c_x = k_x c_{xx} + k_y c_{yy} \text{ on } -2 \le x \le 3, \ -2 \le y \le 2, \ 0 \le t \tag{3.32}$$

with initial data

$$c_0(x,y) = \begin{cases} 1 - R & \text{if } 0 \le x \le 1 \\ 0 & \text{otherwise} \end{cases} \tag{3.33}$$

where $R = \text{sqrt}(x^2 + y^2)$. The boundary condition at time t is

$$c(x,y,t) = 0 \tag{3.34}$$

when $x = -2$, or $x = 3$, or $y = \pm 2$.

The stability condition for this method is $k_x\mu \le 1/4$. The stability analysis is similar to that done on the one- dimensional equation above, i.e. by replacing c_m^n with $g^n e^{im\theta}$ in equation (3.30) and solving for g gives

$$g = 1 - 4k_x\mu\sin^2\frac{1}{2}\theta - 4k_y\mu\sin^2\frac{1}{2}\theta - i\bar{u}\lambda\sin\theta \tag{3.35}$$

or since $k_x = k_y$

$$g = 1 - 8k_x\mu \sin^2 \frac{1}{2}\theta - i\bar{u}\lambda \sin\theta. \tag{3.36}$$

Using the condition $|g| \leq 1$ from Theorem 2.2.1, Theorem 2.2.3 and Corollary 2.2.2 in Strikwerda (26), which shows that the first derivative term can be ignored, gives $8k_x\mu \sin^2 \frac{1}{2}\theta \leq 2$ so $k_x\mu \leq 1/4$. Thus this method is conditionally stable. The results of using this finite difference scheme to solve equation (3.29) will be presented in Section 4.3.3.

3.4.4 The 3-D advection-diffusion equation: The three–dimensional advection-diffusion equation (3.37) includes diffusion in the downwind (x-axis), crosswind (y-axis) and vertical (z-axis) directions. Advection is assumed to be negligable in the crosswind and vertical directions since the wind direction is in the x-axis direction. So again, the only advection term is $\bar{u}c_x$,

$$c_t + \bar{u}c_x = k_x c_{xx} + k_y c_{yy} + k_z c_{zz} \tag{3.37}$$

where $\bar{u}$ is the wind speed (positive constant), and $k_x = k_y = k_z$ are constant in this research and are the x-axis, y-axis and z-axis diffusivity terms respectively. In this research two variations of three-dimensional equation are examined. One variation uses a forward-time central-space finite difference scheme to solve equation (3.37) with the condition at $t = 0$

$$c(x, y, z, 0) = 80 \quad \text{for} \quad x = y = z = 0 \tag{3.38}$$

where 80 was chosen from example problem 2 described above. The boundary condition at time t is

$$c(x, y, z, t) = 0 \tag{3.39}$$

when $x = \pm 10$ or $y = \pm 10$, or $z = \pm 10$. The forward-time central-space finite difference scheme used to solve equation (3.37) is

$$\begin{aligned} \frac{c_{i,j,k}^{n+1} - c_{i,j,k}^{n}}{\Delta t} + \bar{u}\frac{c_{i+1,j,k}^{n} - c_{i-1,j,k}^{n}}{2\Delta x} = \quad & (k_x/\Delta x^2)(c_{i+1,j,k}^{n} - 2c_{i,j,k}^{n} + c_{i-1,j,k}^{n}) \\ & + (k_y/\Delta y^2)(c_{i,j+1,k}^{n} - 2c_{i,j,k}^{n} + c_{i,j-1,k}^{n}) \\ & + (k_z/\Delta z^2)(c_{i,j,k+1}^{n} - 2c_{i,j,k}^{n} + c_{i,j,k-1}^{n}) \end{aligned} \tag{3.40}$$

which is equivalent to

$$\begin{aligned} c_{i,j,k}^{n+1} = \quad & c_{i,j,k}^{n} - 1/2\bar{u}\lambda(c_{i+1,j,k}^{n} - c_{i-1,j,k}^{n}) \\ & + k_x\alpha(c_{i+1,j,k}^{n} - 2c_{i,j,k}^{n} + c_{i-1,j,k}^{n}) \\ & + k_y\beta(c_{i,j+1,k}^{n} - 2c_{i,j,k}^{n} + c_{i,j-1,k}^{n}) \\ & + k_z\gamma(c_{i,j,k+1}^{n} - 2c_{i,j,k}^{n} + c_{i,j,k-1}^{n}) \end{aligned} \tag{3.41}$$

where $c_{i,j,k}^{n} = c(x_i, y_j, z_k, t_n) = c(i\Delta x, j\Delta y, k\Delta z, n\Delta t)$, $\lambda = \Delta t/\Delta x$, $\alpha = \Delta t/\Delta x^2$, $\beta = \Delta t/\Delta y^2$, and $\gamma = \Delta t/\Delta z^2$.

The solution to this scheme will be compared to the exact analytical solution from Seinfeld (25:536) which is

$$\begin{aligned} c(x, y, z, t) = \quad & [s/(8(\pi t)^{3/2}(k_x k_y k_z)^{1/2})] \\ & \cdot \exp[-(x - \bar{u}t)^2/(4k_x t) - y^2/(4k_y t) - z^2/(4k_z t)] \end{aligned} \tag{3.42}$$

where s is the source term, or 80 in this case.

The other variation uses a forward-time central-space finite difference scheme to solve equation (3.43) which adds a source term, or forcing function, $\delta(t) \cdot f(x,y,z)$ to equation (3.37) instead of initial conditions which is

$$c_t + \bar{u}c_x = k_x c_{xx} + k_y c_{yy} + k_z c_{zz} + \delta(t) \cdot f(x,y,z). \tag{3.43}$$

In equation (3.43)

$$f(x,y,z) = \sin\frac{(x+10)\pi}{20} \cdot \sin\frac{(y+10)\pi}{20} \cdot \sin\frac{(z+10)\pi}{20} \tag{3.44}$$

when $x = y = 0$, and $z =$ height, where height is the height of the source, and $\delta(t)$ is the delta function.

The forward-time central-space finite difference scheme used to solve equation (3.43) is

$$\begin{aligned}\frac{c^{n+1}_{i,j,k} - c^n_{i,j,k}}{\Delta t} + \bar{u}\frac{c^n_{i+1,j,k} - c^n_{i-1,j,k}}{2\Delta x} = \quad & (k_x/\Delta x^2)(c^n_{i+1,j,k} - 2c^n_{i,j,k} + c^n_{i-1,j,k}) \\ & + (k_y/\Delta y^2)(c^n_{i,j+1,k} - 2c^n_{i,j,k} + c^n_{i,j-1,k}) \\ & + (k_z/\Delta z^2)(c^n_{i,j,k+1} - 2c^n_{i,j,k} + c^n_{i,j,k-1}) \\ & + \delta(t) \cdot f(x,y,z)\end{aligned} \tag{3.45}$$

which is equivalent to

$$\begin{aligned}c^{n+1}_{i,j,k} = \quad & c^n_{i,j,k} - 1/2\bar{u}\lambda(c^n_{i+1,j,k} - c^n_{i-1,j,k}) \\ & + k_x\alpha(c^n_{i+1,j,k} - 2c^n_{i,j,k} + c^n_{i-1,j,k}) \\ & + k_y\beta(c^n_{i,j+1,k} - 2c^n_{i,j,k} + c^n_{i,j-1,k}) \\ & + k_z\gamma(c^n_{i,j,k+1} - 2c^n_{i,j,k} + c^n_{i,j,k-1})\end{aligned}$$

$$+ \delta(t) \cdot f(x,y,z) \cdot \Delta t \tag{3.46}$$

where $c^n_{i,j,k} = c(x_i, y_j, z_k, t_n) = c(i\Delta x, j\Delta y, k\Delta z, n\Delta t)$, $\lambda = \Delta t/\Delta x$, $\alpha = \Delta t/\Delta x^2$, $\beta = \Delta t/\Delta y^2$, $\gamma = \Delta t/\Delta z^2$, and $f(x,y,z)$ is the forcing function or source term.

The solution to this scheme will be compared to the exact analytical solution which is

$$\begin{aligned} c(x,y,z,t) = \; & \exp[\omega t] \cdot \sin[\pi(x + \bar{u}t + 10)/20] \\ & \cdot \sin[\pi(y+10)/20] \cdot \sin[\pi(z+10)/20] \end{aligned} \tag{3.47}$$

where $\omega = -(\pi/400)(k_x + k_y + k_z)$.

The stability condition for these methods is $k_x\mu \leq 1/8$, when $k_x = k_y = k_z$. The stability analysis is similar to that done on the two-dimensional equation above, with

$$g = 1 - 16k_x\mu \sin^2 \frac{1}{2}\theta - ia\lambda \sin\theta. \tag{3.48}$$

Using the condition $|g| \leq 1$ from Theorem 2.2.1, Theorem 2.2.3 and Corollary 2.2.2 in Strikwerda (26), which shows that the first derivative term can be ignored, gives $16k_x\mu \sin^2 \frac{1}{2}\theta \leq 2$ and $k_x\mu \leq 1/8$. Thus this method is also conditionally stable. The results of using this finite difference scheme to solve equations (3.37) and (3.43) will be presented in Section 4.3.4.

3.4.5 Steady state equation. This subsection describes the steady state two dimensional advection-diffusion equation and the numerical method used to solve it in this research. Following reference (25), if $c_t \to 0$ as $t \to \infty$ and k_x is negligible in equation (3.37), then

$$\bar{u}c_x = k_y c_{yy} + k_z c_{zz} \tag{3.49}$$

with initial conditions

$$\begin{aligned} c(0,y,z) &= \sin(\pi(y+10)/20) \cdot \sin(\pi(z+10)/20) \\ c(x,y,z) &= 0 \quad y,z \rightarrow \pm 10 \end{aligned} \tag{3.50}$$

where $\bar{u}$ is the wind speed (positive constant), $k_y = k_z$ both equal $\bar{u}$ in this research and are the y-axis and z-axis diffusivity terms respectively, and q is the source term. The central-space finite difference scheme used to solve equation (3.49) is

$$\begin{aligned} \bar{u}\frac{c^n_{m+1,p} - c^n_{m-1,p}}{2\Delta x} = \quad & (k_y/\Delta y^2)(c^n_{m+1,p} - 2c^n_{m,p} + c^n_{m-1,p}) \\ & + (k_z/\Delta z^2)(c^n_{m,p+1} - 2c^n_{m,p} + c^n_{m,p-1}). \end{aligned} \tag{3.51}$$

The solution to equation (3.51) will be compared to the exact solution which is

$$c = \exp[-\frac{\pi^2}{200}] \cdot \sin(\frac{\pi(y+10)}{20}) \cdot \sin(\frac{\pi(z+10)}{20}). \tag{3.52}$$

The stability condition analysis is similar to the two-dimensional advection-diffusion equation discussed in Section 3.4.3.

3.5 Conclusion

Three Gaussian plume models are described in this chapter, SCREEN, AFTOX, and GAUSPLUM. SCREEN uses a form that includes reflection terms, but that doesn't include crosswind terms. SCREEN, therefore, mainly does calculations of pollutant concentrations under the centerline of the plume. AFTOX uses three forms of the Gaussian model which include two Gaussian puff models, one when there is no inversion in the atmosphere, and one when there is an inversion. AFTOX also uses the Gaussian plume model. GAUSPLUM uses the Gaussian plume model. The

results and comparison of SCREEN, AFTOX, the program GAUSPLUM developed in this study, will be presented in Section 4.2. The other models of air pollution transport discussed in this chapter all follow from the three-dimensional advection-diffusion equation. They are all special cases of the three-dimensional advection-diffusion equation. The advection equation does not include any diffusion. The one- and two-dimensional advection-diffusion equations only include the downwind, and downwind and crosswind diffusion terms, respectively. The steady-state equation has no change in the concentration with respect to time. The results of the numerical schemes used to solve the advection equation, one, two and three dimensional advection-diffusion equations, and the two- dimensional steady state equation will be discussed in Sections 4.3.1, 4.3.2, 4.3.3, 4.3.4, and 4.3.5 respectively.

IV. Results

4.1 Introduction

The purpose of this study is to examine models which use analytical solutions in their calculations of pollutant concentrations and to find numerical solutions of the various partial differential equations used in pollution modeling. This chapter includes a comparison of the models SCREEN (10), AFTOX (19), and the program GAUSPLUM (developed in this study) using the examples taken from Turners workbook (27) described in Section 3.3. The results of the numerical solutions of the different equations described in Section 3.4 are also presented here.

The comparison of SCREEN, AFTOX, and GAUSPLUM will be discussed in Section 4.2 The results of the numerical schemes used to solve the partial differential equations and the sections they are discussed as follows: the advection equation in Section 4.3.1, the one, two and three dimensional advection-diffusion equations will be discussed in Sections 4.3.2, 4.3.3, and 4.3.4, respectively, and two-dimensional steady state equation in Section 4.3.5.

4.2 Comparison of Gaussian models

A comparison of the three Gaussian models which use analytical solutions in their calculations is done using the six problems from Turner's workbook (27) described in Section 3.3. Table 4.1 shows the comparison of SCREEN, AFTOX, GAUSPLUM using problems 1, 2, and 3, which calculate the concentration at a certain location where a receptor is located. Table 4.2 shows the comparison of SCREEN, AFTOX, GAUSPLUM using problems 4,5, and 11, which give the location and value of the maximum concentration. The models all have solutions to the six problems within an order of magnitude of the solutions given in Turner's workbook. This is expected since they each use a form of the Gaussian plume model.

Turner Workbook Examples (27)				
ex. #	Turner (g/m^3)	SCREEN (g/m^3)	AFTOX (g/m^3)	GAUSPLUM (g/m^3)
1	11.0 E-6	11.34 E-6	4.0 E-6	10.95 E-6
2	33.0 E-6	41.45 E-6	23.0 E-6	32.9 E-6
3	13.0 E-6	NA	9.0 E-6	12.95 E-6

Table 4.1 Comparison of concentrations at given point

As Table 4.1 shows, the SCREEN model produces higher estimates than Turner's workbook results for problems 1 and 2 and isn't applicable for problem 3. This result is because the equation (see eq. 3.1) used in the model lacks the term $\exp[-\frac{1}{2}(y/\sigma_y)^2]$ found in both AFTOX and GAUSSPLUM which allows the receptor to be at points away from the x-axis. The AFTOX model consistently underestimated the values for each of the three problems in Table 4.1. All three models results are within an order of magnitude on either side of the results in Turners workbook.

The results are similar for the problems which determine the maximum concentration and its location. Table 4.2 shows these results. Again all three models solutions are within an order of magnitude of the problems in Turners workbook.

Turner Workbook Examples (27)				
ex. #	Turner (g/m^3) at m	SCREEN (g/m^3) at m	AFTOX (g/m^3) at m	GAUSPLUM (g/m^3) at m
4	280.0 E-6 at 1000	235.6 E-6 at 1005	180.0 E-6 at 1420	263.4 E-6 at 1000
5	1.1 E-4 at 5600	0.782 E-4 at 5454	1.0 E-4 at 2839	1.13 E-4 at 5475
11	6.4 E-5 at 13000	2.52 E-5 at 12434	6.8 E-5 at 6109	6.08 E-5 at 12500

Table 4.2 Comparison of maximum concentration calculations

As described in Chapter III both the SCREEN (Section 3.2.1) and AFTOX (Section 3.2.2) models require additional input such as ambient temperature (outside air temperature) and exit velocity of the pollutant. Figure 4.1 is output from the SCREEN program and Figure 4.2 is from the AFTOX program for problem 2 in Turners workbook. The following is a brief description of the two figures. Each program asks for ambient air temperature, emmission rate, wind speed, stack height,

stack gas temperature, and the location of the receptor for the calculation. Problem 2 says it is an overcast winter morning. SCREEN asks for the ambient temperature only, while AFTOX asks for both the ambient temperature and the time and date of the calculation. The date chosen for AFTOX for this problem is January 6, 1992. For these programs the value of the ambient temperature is taken from climatological data for the WPAFB area for the time of year described in each problem. The exit velocity and stack diameter used here come from other examples in the workbook.

The program GAUSPLUM simply asks for the stability class, emission rate, wind speed, stack height and the location of the receptor. The output for GAUSPLUM displays the value of the distance and the concentration value. It only does calculations for one distance at a time so for problems 4, 5, and 11 this program is run several times to determine the maximum concentration and its distance.

```
 *** SCREEN-1.1 MODEL RUN ***
 *** VERSION DATED  88300 ***
SIMPLE TERRAIN INPUTS:
   SOURCE TYPE            =    POINT
   EMISSION RATE (G/S)    =    80.00
   STACK HEIGHT (M)       =    60.00
   STK INSIDE DIAM (M)    =     2.00
   STK EXIT VELOCITY (M/S)=     2.00
   STK GAS EXIT TEMP (K)  =   293.50
   AMBIENT AIR TEMP (K)   =   284.00
   RECEPTOR HEIGHT (M)    =      .00
   IOPT (1=URB,2=RUR)     =     2
   BUILDING HEIGHT (M)    =      .00
   MIN HORIZ BLDG DIM (M) =      .00
   MAX HORIZ BLDG DIM (M) =      .00

BUOY. FLUX =     .03 M**4/S**3;  MOM. FLUX =    3.99 M**4/S**2.

*** STABILITY CLASS  4 ONLY ***
*** 10-METER WIND SPEED OF   6.0 M/S ONLY ***
**********************************
*** SCREEN DISCRETE DISTANCES ***
**********************************

 CALCULATION        MAX CONC    DIST TO   TERRAIN
  PROCEDURE        (UG/M**3)    MAX (M)    HT (M)
--------------    -----------   -------   -------
SIMPLE TERRAIN      41.45          500.        0.
```

Figure 4.1 Partial output of SCREEN program for problem 2

4.3 Results and comparison of exact solutions

This section shows the results of the finite difference methods used to solve the partial differential equations described in chapter III.

```
        USAF TOXIC CHEMICAL DISPERSION MODEL
                      AFTOX
WPAFB OH
DATE: 01-06-1992
TIME: 0800 LST

CONTINUOUS BUOYANT PLUME

TEMPERATURE = 6 C
WIND DIRECTION = 360
WIND SPEED = 6 M/S
SUN ELEVATION ANGLE IS 5 DEGREES
CLOUD COVER IS 8 EIGHTHS
CLOUD TYPE IS LOW (St, Ns, FOG)
GROUND IS DRY
THERE IS NO INVERSION
ATMOSPHERIC STABILITY PARAMETER IS 3.5
EMISSION RATE(KG/MIN) = 4.8
EFFLUENT IS STILL BEING EMITTED
STACK HEIGHT ABOVE GROUND(M) = 60
GAS STACK TEMP(C) = 15
VOLUME FLOW RATE(M3/MIN) = 4
EFFECTIVE PLUME HEIGHT(M) = 60 AT DISTANCE(M) = 2
CONCENTRATION AVERAGING TIME IS 10 MIN
HEIGHT ABOVE GROUND IS 0 M
DOWNWIND DISTANCE IS 500 M
CROSSWIND DISTANCE IS 0 M
-------------------------------------------------
THE CONCENTRATION IS .018 MG M-3

-------------------------------------------------
```

Figure 4.2 Partial output of AFTOX program for problem 2

4.3.1 Advection equation. The numerical solution to the advection equation (see eq. 3.11) is found using the Lax-Friedrichs and the leapfrog finite difference schemes. For both schemes $\lambda = 0.8$, $\Delta x = 0.1$, and $\Delta t = 0.08$. On the boundary, when $x = -2$, $c = 0$, and $c_m^{n+1} = c_{m-1}^{n+1}$ when $m = 3$. The finite difference formula, equation (3.17), in Section 3.4.1 is used in the Ada program in appendix (B.1) in procedure Compute_New. Figure 4.3 shows the solution to the initial-boundary value problem (see eq. 3.12) discussed in Section 3.4.1 using the Lax-Friedrichs finite difference scheme. Figure 4.4 shows the solution to the same initial-boundary value problem using the leapfrog finite difference scheme, equation (3.18), in procedure Compute_New.

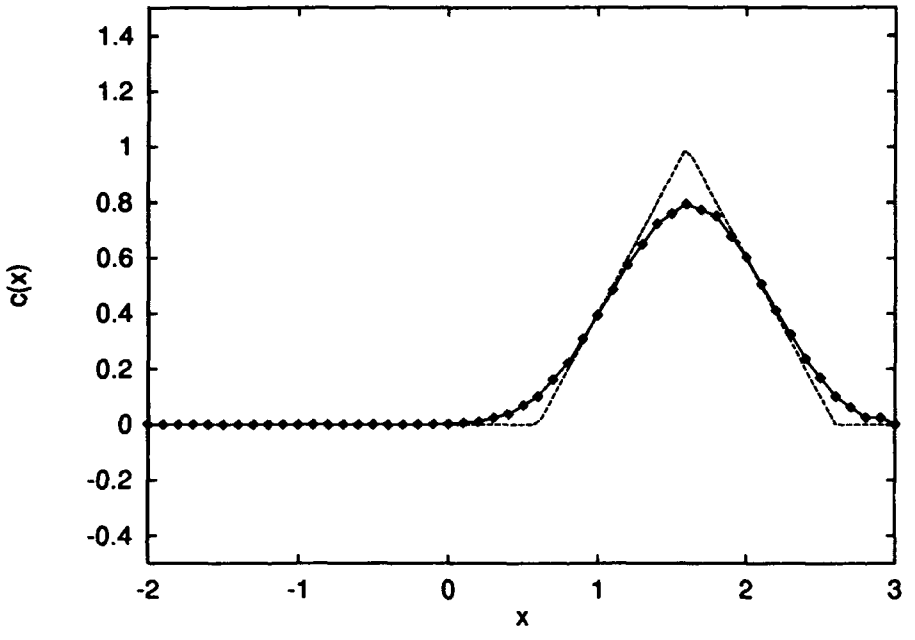

Figure 4.3 Lax-Friedrichs solution of advection equation

In both Figure 4.3 and Figure 4.4 the exact solution is shown as a dashed line, and the solution of the finite difference scheme is shown as the curve with diamonds. The leapfrog method has more oscillations in its solution than the Lax-Friedrichs, but the overall accuracy is better for the leapfrog scheme. The approximation at the peak in Figure 4.4, the leapfrog solution, is much better than in Figure 4.3. The

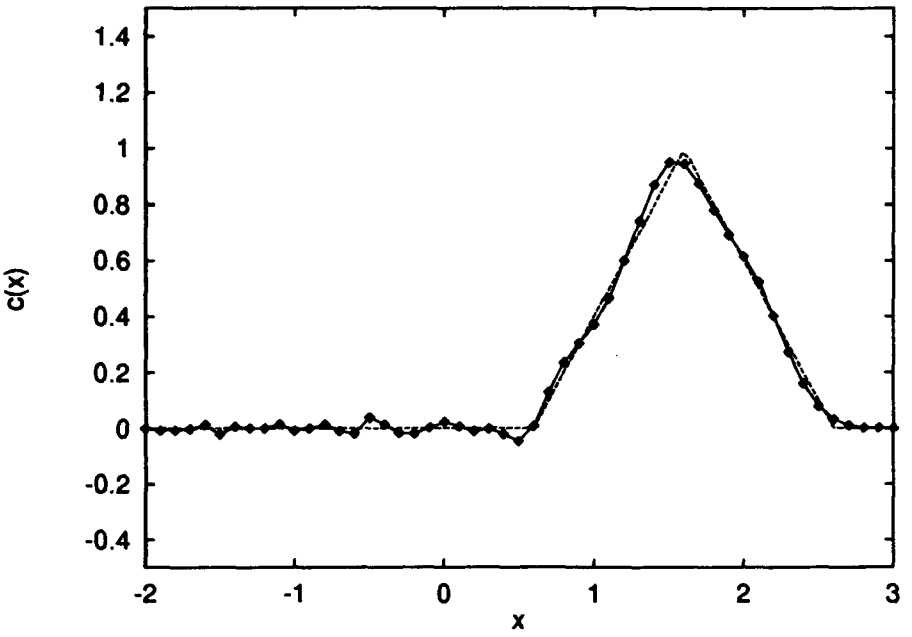

Figure 4.4 Leapfrog solution of advection equation

solution of the Lax-Friedrichs scheme can be improved by decreasing the value of Δx while keeping the same value of λ.

As discussed in Section 3.4.1 the Lax-Friedrichs scheme is conditionally stable. To show this $\lambda = 1.6$ is used in the scheme with the results shown in Figure 4.5. In Figure 4.5 the exact solution is shown as a solid line while the solution of the Lax-Friedrichs scheme is shown as a line with diamonds.

Figures 4.3 and 4.4 show that the solutions of the Lax-Friedrichs scheme and the leapfrog scheme are reasonable approximations to the solution of the advection equation. As the values of Δx and Δt are decreased, while keeping λ constant, the solutions of the schemes become better approximations to the advection equation. In the next section a diffusion term is added to the advection equation and the one-dimensional advection-diffusion equation is looked at.

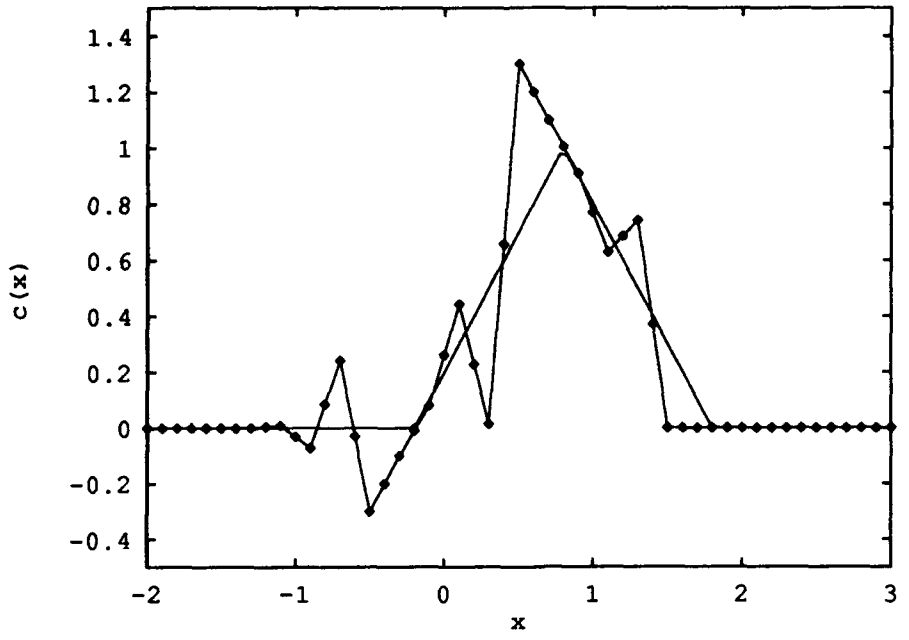

Figure 4.5 Lax-Friedrichs solution with $\lambda = 1.6$

4.3.2 1-D advection-diffusion equation. The one-dimensional advection-diffusion equation (see eq. 3.22) is a combination of the advection equation described in Section 4.3.1 and the diffusion equation discussed in Appendix C.1. The forward-time central-space scheme (see eq. 3.23) is used to solve the advection-diffusion equation (see eq. 3.22). It is in Ada procedure Compute_New which is found in Appendix B.2.

The stability condition for this method, discussed in Section 3.4.2, is $k_x\mu \leq 1/2$ where $mu = \Delta t/\Delta x^2$. This stability condition means the time step Δt is at most $\Delta x^2/2k_x$ and when decreasing Δx by half to increase the spatial accuracy, Δt must decrease by one-fourth. This restriction limits practical use of this method. The data in Table 4.3 show the cases used in this method.

Table 4.3 includes: $\bar{u}$, the advection coefficient; k_x, the diffusion coefficient; Δx, the space incrementer; Δt, the time incrementer; the range in space; $k_x\mu$, which from the stability analysis must be less than or equal to one half; $\alpha = (\Delta x\bar{u})/(2k_x)$,

Try #	$\bar{u}$	k_x	Δx	Δt	x-range	$(k_x\Delta t)/\Delta x^2$ (≤ 0.5)	$\alpha = (\Delta x\bar{u})/(2k_x)$ (≤ 1.0)	t
1	4	4	1	0.1	-10..10	0.4	0.5	2.0
2	10	4	1	0.1	-10..10	0.4	1.25	2.0
3	8.1	4	1	0.1	-10..10	0.4	1.0125	2.0
4	6	7	2	0.2	-20..20	0.35	0.85714	10.0
5	1	1	0.5	0.05	-10..10	0.2	0.25	2.0
6	3.95	1	0.5	0.05	-10..10	0.2	0.9875	2.0
7	4.01	1	0.5	0.05	-10..10	0.2	1.0025	2.0
8	1.0025	0.25	0.5	0.05	-10..10	0.05	1.0025	2.0
9	1.0025	0.25	0.5	0.2	-10..10	0.2	1.0025	2.0
10	4	0.25	0.5	0.25	-10..10	0.15	4	2.0

Table 4.3 1-D advection-diffusion test data

which Strikwerda shows must be less than or equal to one; and t, the time of final calculation.

The cases included in Table 4.3 all satisfy the stability condition $k_x\mu \leq 1/2$. These cases were done to test the other condition $\alpha = (\Delta x\bar{u})/(2k_x) \leq 1$ to see if oscillations detracted from the accuracy of the solution. The cases also varied the advection term to see its impact on the solutions. In Figure 4.6 the solution for test case number 6 is plotted.

In Figure 4.7 the solution for test case number 10 is plotted. Case number 10 has an oscillating solution because the $\alpha \leq 1$ condition is not met. In this case $\alpha = 4.0$ which causes the oscillation even though the stability condition is met.

These two cases, numbers 6 and 10 in Table 4.3, have similar advection terms, 3.95 and 4.0, respectively. The diffusion terms differ by a larer margin and contribute to the differences in the α condition and also to the oscillations in case 10. Case 6 satisfies the α-condition and has a non-oscillating solution, while case 10 violates the condition and has an oscillating solution.

In the other cases of Table 4.3 the advection coefficient and the diffusion coefficient are varied to see the effect of a higher advection term than diffusion term. In

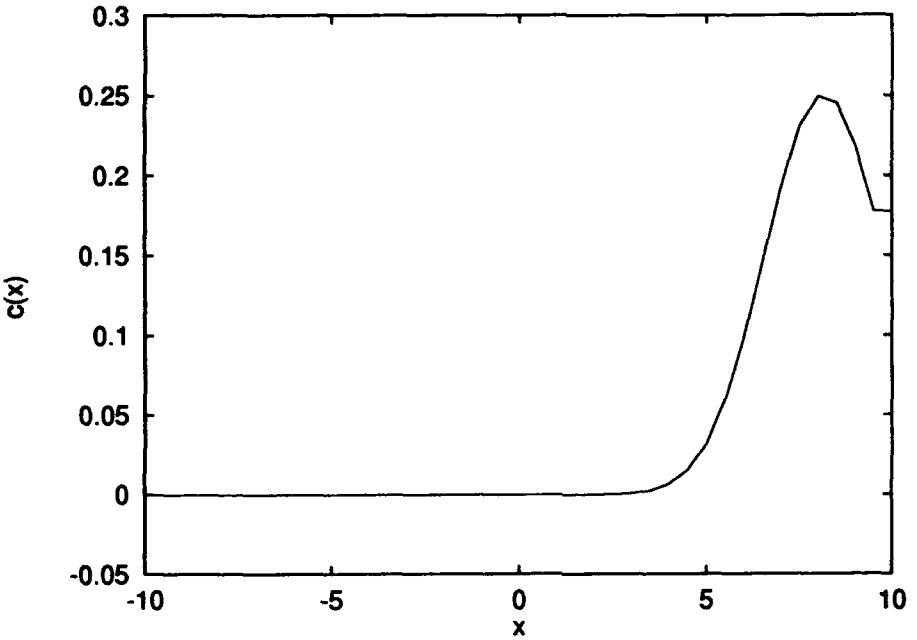

Figure 4.6 Test case 6

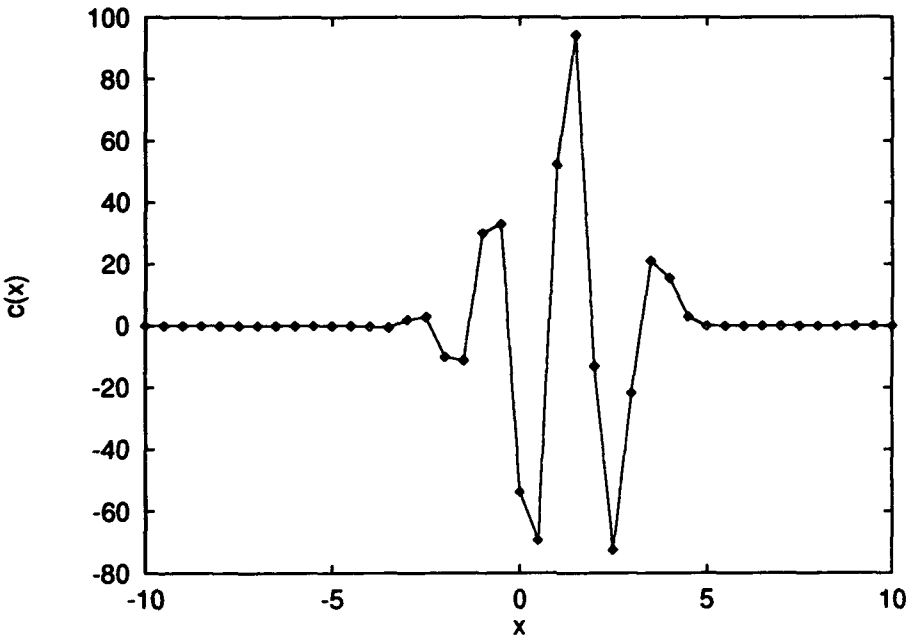

Figure 4.7 Test case 10

all of the cases with α greater than one the solution oscillates as expected. In the other cases, where α is less than one and the advection coefficient is greater than the diffusion coefficient, the solution is similar to case number six in Figure 4.6. In the next section, a second diffusion coefficient is added to the equation to see if the diffusion has a larger impact on the solution.

4.3.3 2-D advection-diffusion equation. The two-dimensional advection-diffusion equation (see eq. 3.29) adds the crosswind (y-axis) diffusion term to the one-dimensional equation. The forward-time central-space scheme, equation (3.30), in Section 3.4.3 is used to solve the 2-D advection-diffusion equation (3.29). It is used in Ada procedure Compute_Final found in Appendix B.3. The data in Table 4.4 show the cases used in this method.

Try #	$\bar{u}$	k_x	k_y	Δx	Δy	Δt	time
1	2	0.0625	0.0625	0.2	0.2	0.02	1.6
2	0.5	0.0625	0.0625	0.2	0.2	0.02	1.6
3	4	0.05	0.05	0.2	0.2	0.02	1.6
4	4	0.05	0.05	0.1	0.1	0.08	1.6
5	0.5	0.5	0.5	0.2	0.2	0.02	1.6
6	0.5	1	1	0.2	0.2	0.1	1.6
7	0	1	1	0.2	0.2	0.1	1.6
8	0	1	1	0.2	0.2	0.08	1.6
9	0.5	1	1	0.2	0.2	0.04	1.6
10	0.5	0.9	0.9	0.2	0.2	0.04	1.6
11	0.5	1	1	0.2	0.2	0.01	1.6
12	0.5	1	1	0.2	0.2	0.02	1.6
13	1	1	1	0.2	0.2	0.01	1.6
14	4	1	1	0.2	0.2	0.01	1.6
15	0	1	1	0.2	0.2	0.01	1.6
16	0	0.0625	0.0625	0.2	0.2	0.02	1.6
17	1	0.0625	0.0625	0.2	0.2	0.02	1.6
18	1.5	0.0625	0.0625	0.2	0.2	0.02	1.6

Table 4.4 2-D advection-diffusion test data

Table 4.4 includes: $\bar{u}$, the advection coefficient; k_x, k_y, the x-axis, and y-axis diffusion coefficients; Δx, the x-axis incrementer; Δy, the y-axis incrementer; Δt, the time incrementer; and t, the time of final calculation.

The stability condition for this method, discussed in Section 3.4.3, is $k_x\mu \leq 1/4$ where $\mu = \Delta t/\Delta x^2$. This stability condition means the time step Δt is at most $\Delta x^2/2k_x$ and when decreasing Δx by half to increase the spatial accuracy, Δt must decrease by one-eighth. This restriction limits practical use of this method even more than the one-dimensional case.

Two cases that show the difference between complying with the stability condition and violating it are case 5 and case 12. Case 5 is an example of complying with the stability condition, while case 12 is one that violates the condition. The only differences between the two cases are that for case 5, $k_x = 1/2$ and $k_x\Delta t/\Delta x^2 = 1/4$, while for case 12, $k_x = 1.0$ and $k_x\Delta t/\Delta x^2 = 1/2$.

Figure 4.8 shows the solution to equation (3.32) using test case 5 from Table 4.4, where the view is looking at the xz-plane with the x-axis going from left to right, and the concentration is on the z-axis. The y-axis goes into the page. The lines in the figure are the concentration contours for different values of y similar to that in Figure 4.9 which shows the solution using case 5 with $y = 0$. When $y = 0$ the resulting contour is the maximum concentration contour since it is directly downwind from the source. The minimum concentration contour is near the boundary at $y = \pm 10$. Figure 4.10 shows the solution to equation (3.32) using test case 12 with the same view as Figure 4.8. Case 12 violates the stability condition which results in large oscillations. Again the lines in the figure are the concentration contours for a given value of y. Figure 4.11 shows the solution using case 12 with $y = 0$ and shows the maximum concentration contour.

Most of the cases in Table 4.4 vary the terms $\bar{u}$, k_x, k_y, and time while keeping the Δx and Δy constant. One exception to this is case 3 and case 4 which changes Δx, Δy and Δt, but keeps the other terms constant. Case 3 is stable since $k_x\mu = 1/4$

which meets the stability condition. Case 4 on the other hand does not meet the stability condition and is, therefore, unstable.

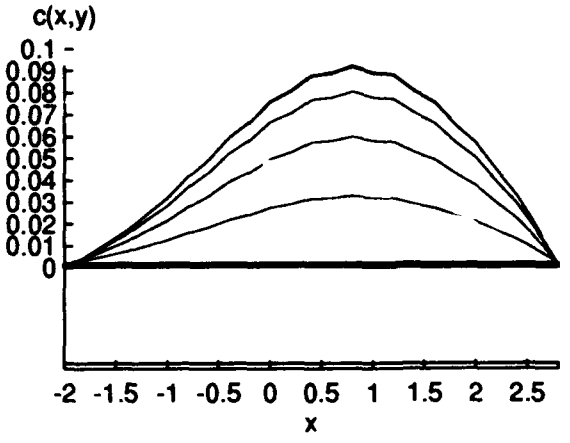

Figure 4.8 2-D test case 5

These results show how only slightly violating the stability condition can cause large oscillations. The same result will be shown in the next section for the 3-D advection-diffusion equation.

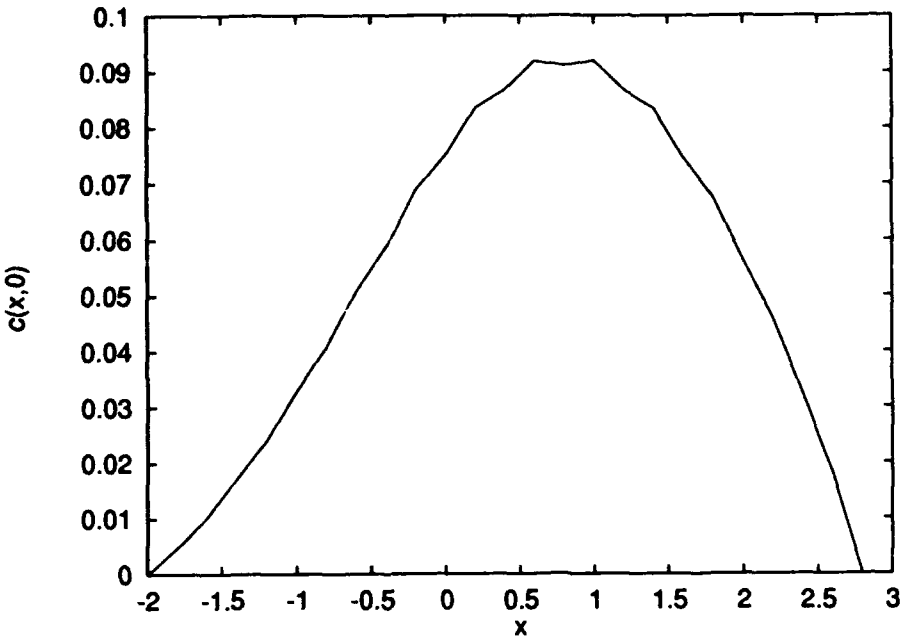

Figure 4.9 2-D test case 5 with maximum concentration

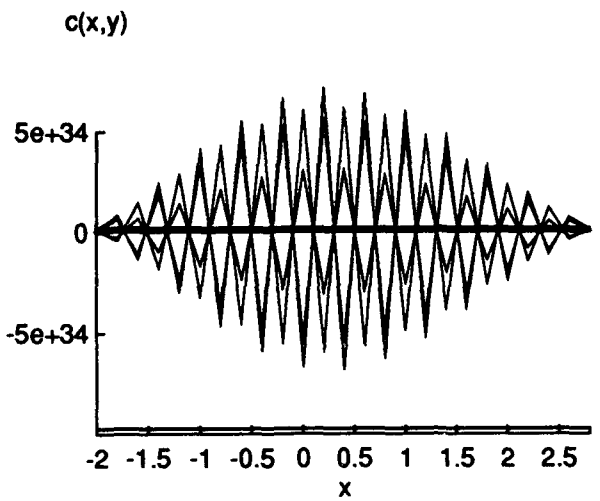

Figure 4.10 2-D test case 12

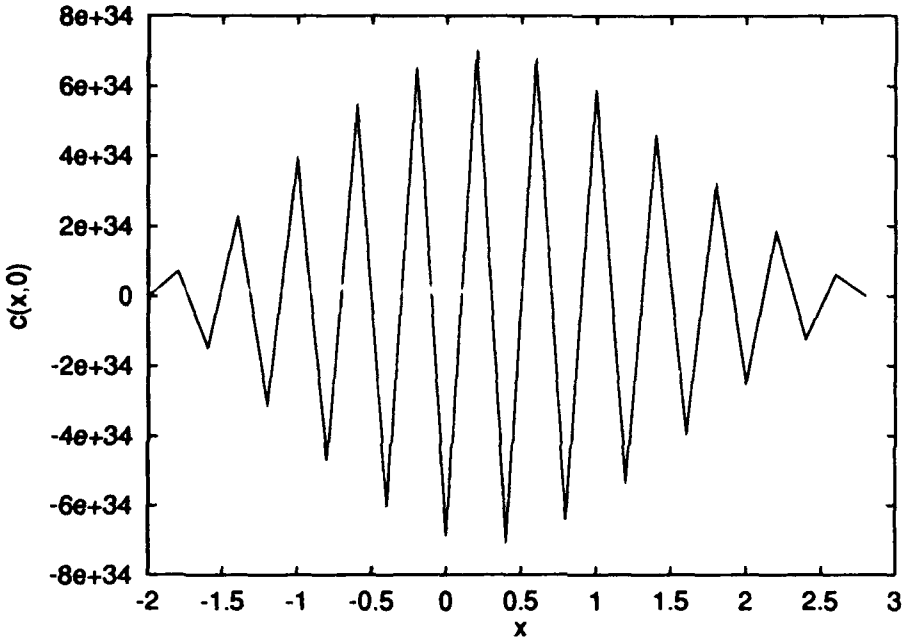

Figure 4.11 2-D test case 12 with $y = 0$

4.3.4 3-D advection-diffusion equation. The three-dimensional advection-diffusion equation (3.37) adds the vertical (z-axis) diffusion term to the two-dimensional equation. The other equations are all special cases of the three-dimensional advection-diffusion equation. The advection equation has no diffusion term. The one-dimensional advection-diffusion equation has no crosswind or vertical diffusion, and the two-dimensional advection-diffusion equation has no vertical diffusion. The steady-state equation has no concentration change with respect to time. This research looks at the three-dimensional advection-diffusion equation in two ways, one with the source as part of the initial conditions, and a second way by adding a forcing function or source term so that initial conditions are incorporated into the equation. This section will look at the results of using a forward-time central-space scheme, equation (3.40) in Section 3.4.4, to solve the three-dimensional advection-diffusion equation with the forcing function added, equation (3.43). The scheme is used in Ada procedure Compute_Final found in Appendix B.4. Table 4.5 shows the

cases used in the method without the source term and with the initial conditions, equation (3.38), discussed in Section 3.4.4.

Case #	$\bar{u}$	k_x	Δt	Δi	stable ≤ 0.125	t
1	0.5	1	1	2	0.25	10
2	0.5	0.5	1	2	0.125	10
3	0.5	0.25	1	2	0.0625	10
4	0.5	0.125	1	2	0.03125	10

Table 4.5 3-D without source term

In Table 4.5: $\bar{u}$ is the advection coefficient, $k_x = k_y = k_z$ are the diffusivity coefficients, Δt is the change in time, Δi is the change in the spatial directions where i is x, y, or z, stable is the stability conditon, and t is the time of the calculation. In these cases the only difference between them is the change in Δt which affects the stability condition. The three previous sections show that the α-condition must be met for the solution to be well-behaved. In this section the stability condition is examined, that is $k_x \Delta t / \Delta i^2 \leq 0.125$, where i is x, y, or z.

Figure 4.12 shows case 3 from Table 4.5, as a line with squares, compared to the exact solution, equation (3.42), as a solid line. The figure shows the solution for $x = 8, \quad z = 0, \quad$ and $t = 10$.

Figure 4.13 shows cases 1, 2 and 3 from Table 4.5, as a line with crosses, a line with diamonds and a line with squares, respectively, compared to the exact solution, equation (3.42), as a solid line. The figure again shows the solution for $x = 8, \quad z = 0, \quad$ and $t = 10$.

The reason the numerical solution only seems to move toward the exact solution near $y = 0$ and not change much near the boundary is that the boundary condition $c(x, y, z, t) = 0$ when $x = \pm 10$ or $y = \pm 10$, or $z = \pm 10$ does not represent what the exact solution does at those boundary points. The exact solution has the same boundary condition, except that the boundary is at $\pm\infty$. The boundary condition for the numerical solution causes the boundary to act as a sink for the pollutant

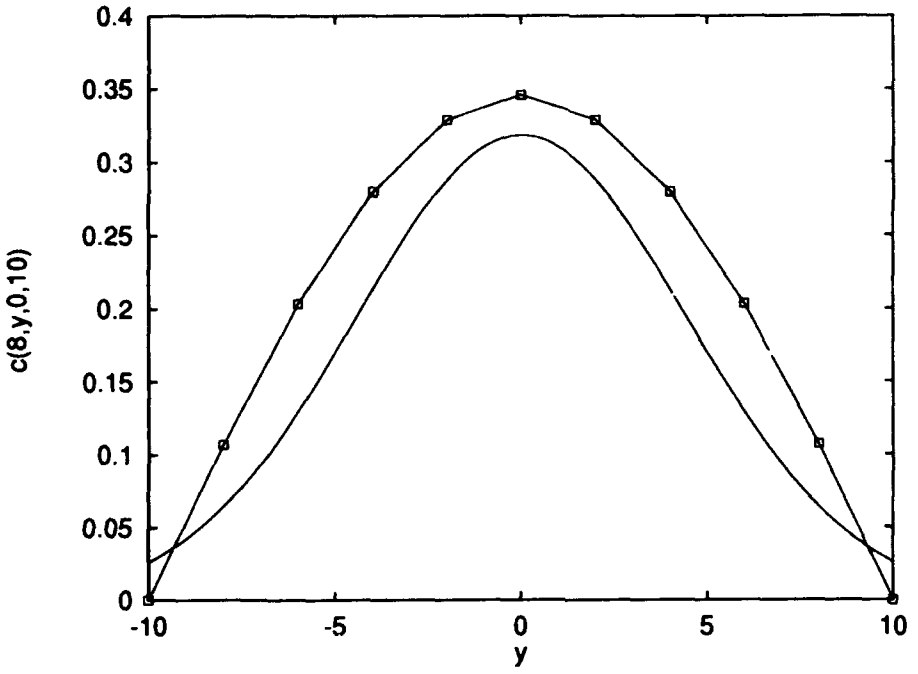

Figure 4.12 3-D test case 3 and exact solution

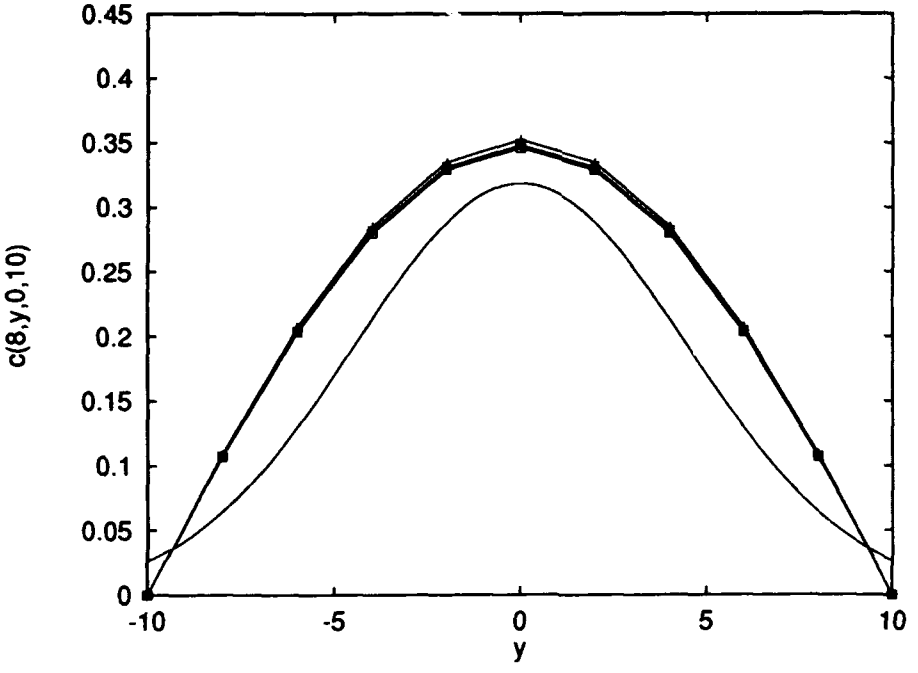

Figure 4.13 3-D case 2,3,4 and exact solution

as it is dispersed and advected. The other variation described in Section 3.4.4 has conditions such that the exact calculation has the same boundary conditions as the numerical calculation.

Table 4.6 shows the cases used in the scheme with the source term, equation (3.45).

Case #	$\bar{u}$	k_x	Δi	stable ≤ 0.125	Δt	t
1	0.2	1	5	0.04	1	10
2	0.2	1	5	0.02	0.5	10
3	0.2	1	5	0.01	0.25	10
4	0.2	1	5	0.005	0.125	10

Table 4.6 3-D with source term

In Table 4.6: $\bar{u}$ is the advection coefficient, $k_x = k_y = k_z$ are the diffusivity coefficients, Δi is the change in the spatial directions where i is x, y, or z, stable is the stability conditon, Δt is the change in time, and t is the time of the calculation. Again, Δt is the only difference between the cases and this changes the stability condition as shown in the table. These cases all satisfy the stability condition. The following three figures are of cases 1, 2, and 3 which show that as Δt is decreased the numerical solution converges on the exact solution. This result would be the same as decreasing both the spatial terms Δi and Δt such that the stability condition remained the same as in Table 4.6. Figure 4.14 shows case 1 as a line with squares. Figure 4.15 shows case 2 as a line with crosses. Figure 4.16 shows case 3 as a line with diamonds. In all three figures the exact solution, equation (3.47), is a solid line.

The two variations of the three-dimensional advection-diffusion equation in this section show how the boundary conditions can have an effect on the numerical solution. The boundary condition for the variation without the source term does not represent the actual conditions where the boundary is at $\pm\infty$. The numerical solution is less than actual since the boundary condition artificially lowered the value

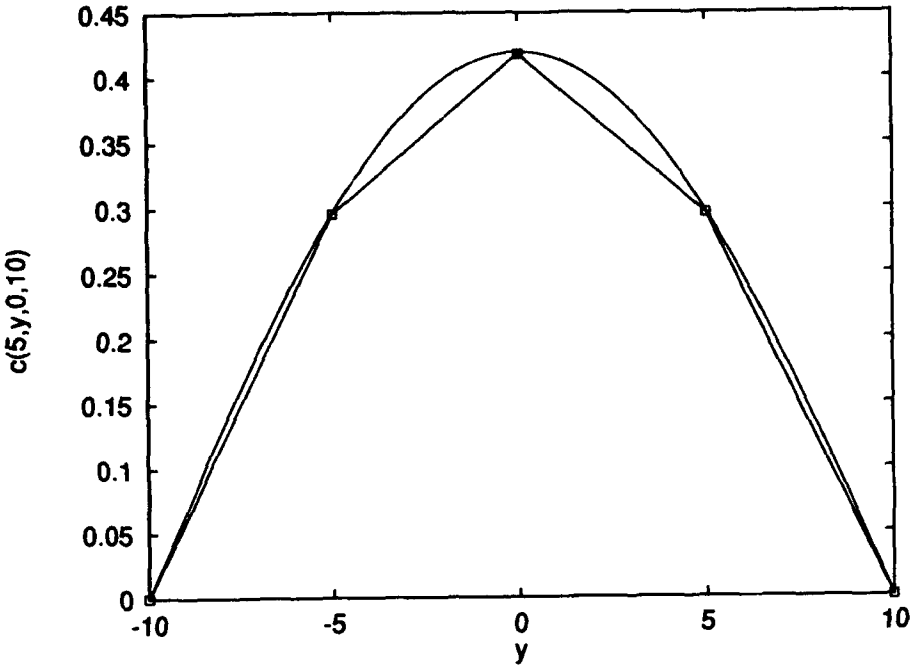

Figure 4.14 3-D case 1 with source term

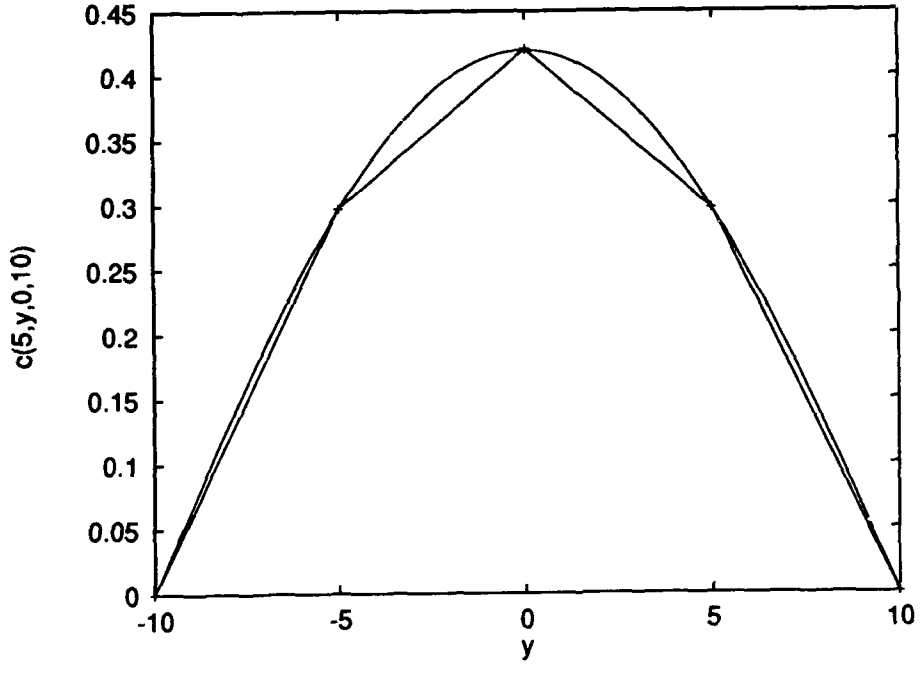

Figure 4.15 3-D case 2 with source term

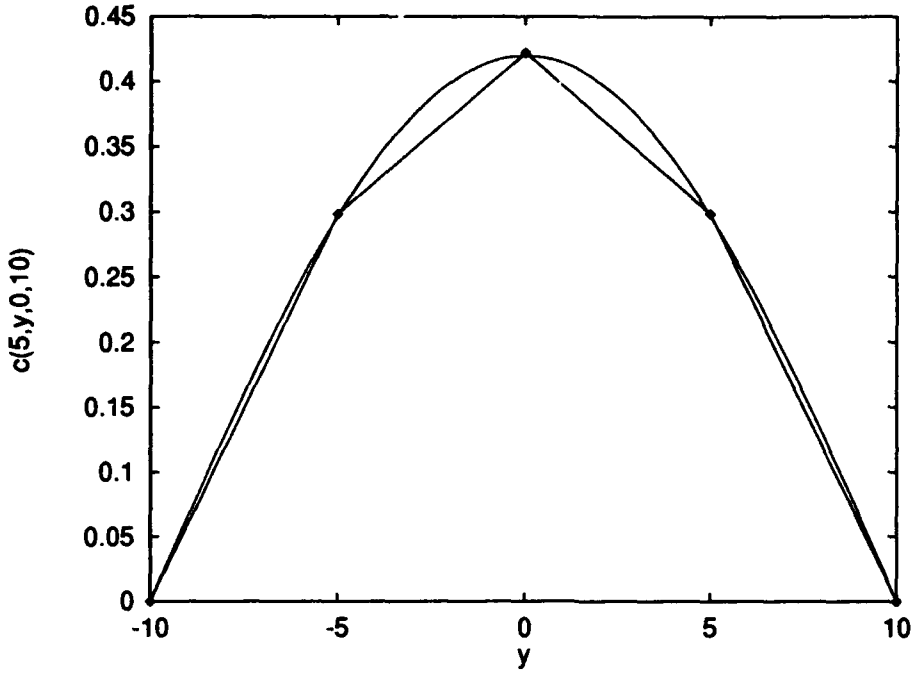

Figure 4.16 3-D case 3 with source term

at points near the boundary. The other variation has the same boundary conditions as the exact calculation and the numerical and exact solutions are much closer.

4.3.5 Steady state equation. The two-dimensional steady state equation is similar to the two-dimensional advection-diffusion equation except the concentration does not change with respect to time. This section will look at the results of using a central-space scheme, equation (3.51) in Section 3.4.5, to solve the two-dimensional steady state equation (3.49) with the forcing function added in the initial conditions, equation (3.50). The scheme is used in Ada procedure Compute_Final found in Appendix B.5. Table 4.7 shows the cases used in this scheme.

In Table 4.7 $\bar{u}$ is the advection coefficient, $k_y = k_z = \bar{u}$ are the diffusivity coefficients, Δx is the change in x-axis direction, Δi is the change in the crosswind and vertical directions where i is y, or z, and stable is the stability conditon $(k_y \Delta x)/(\Delta y^2)$. In these cases the only difference between them is the change in Δx which also changes value of the stability condition by the same factor.

Case #	$\bar{u}$	k_y	Δx	Δi	stable ≤ 0.25
1	4	4	5	5	0.2
2	4	4	2	5	0.08
3	4	4	1	5	0.04
4	4	4	0.5	5	0.02

Table 4.7 2-D steady state data

Figure 4.17 shows case 1 as a line with squares. Figure 4.18 shows case 3 as a line with crosses. Figure 4.19 shows case 4 as a line with diamonds. Figure 4.20 shows a comparison of all four cases with the exact solution. In all four figures the exact solution, equation (3.52), is a solid line.

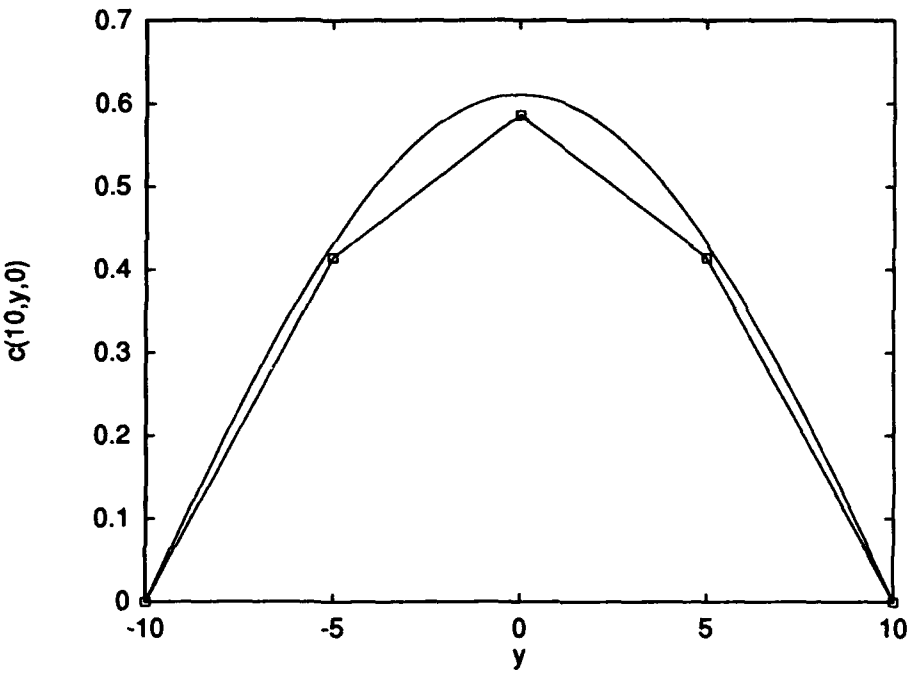

Figure 4.17 Steady state case 1

If the stability condition is satisfied, then decreasing the value of Δx does not require any changes in other variables. However, if the stability condition is satisfied, and a decrease in Δy and Δz is wanted by one half, then Δz must be decreased by one fourth.

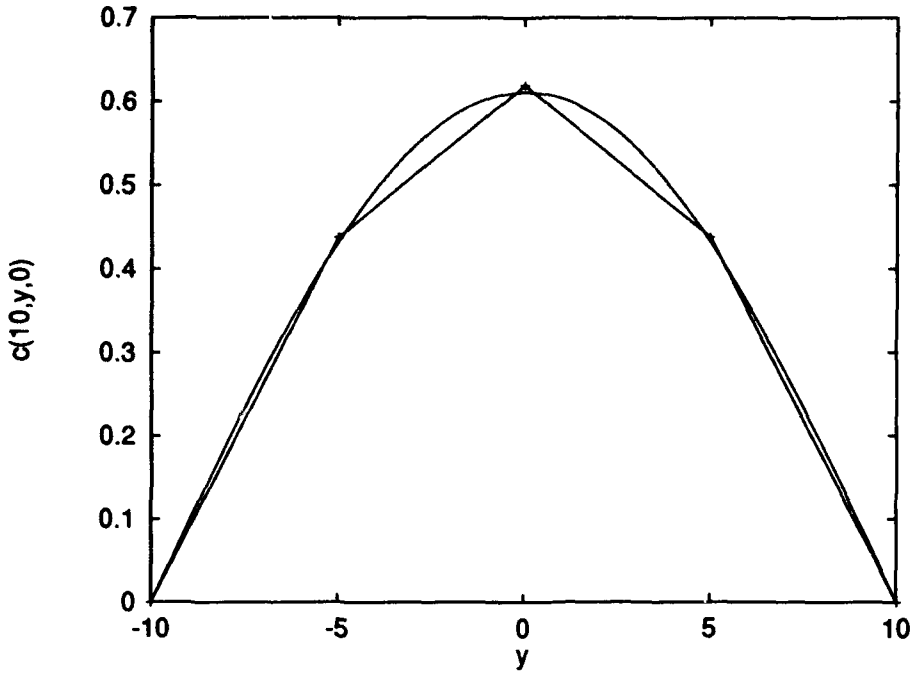

Figure 4.18 Steady state case 3

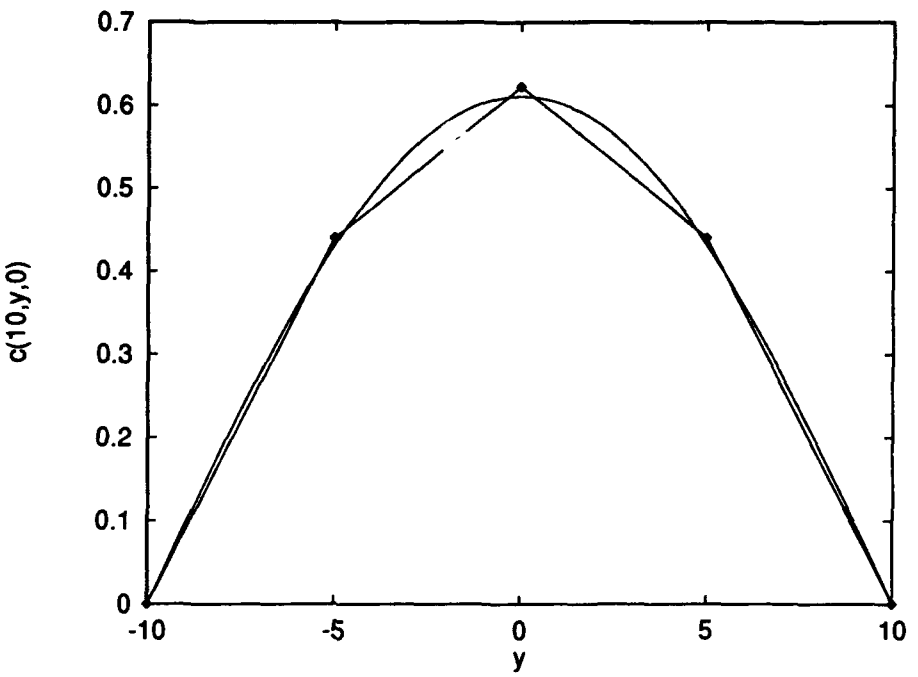

Figure 4.19 Steady state case 4

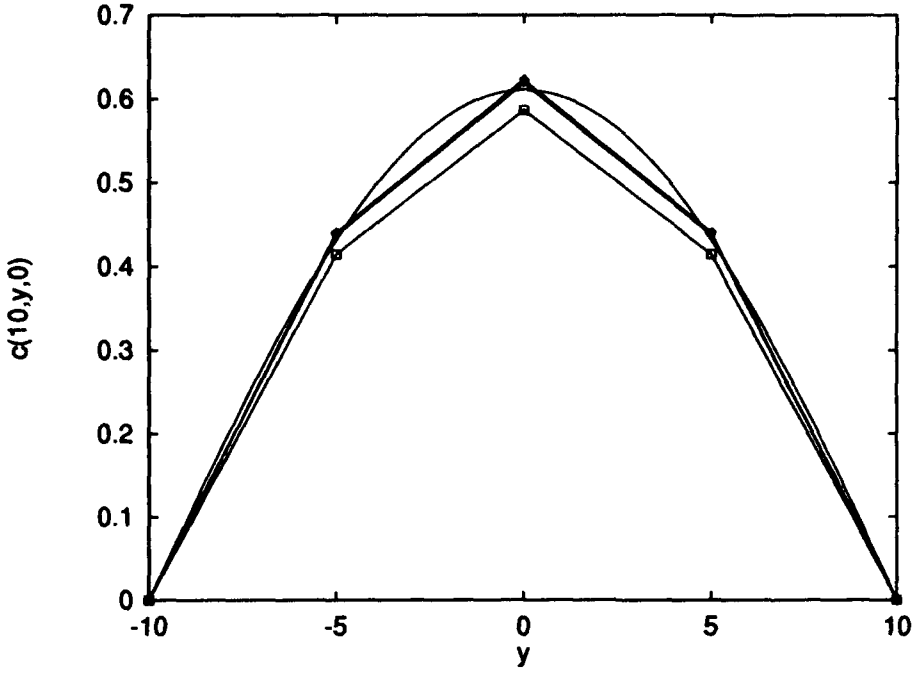

Figure 4.20 Steady state comparison

4.4 Conclusion

The results of the comparison of SCREEN, AFTOX, and GAUSPLUM show that they can be used in general cases such as those in the problems of Turner's workbook. SCREEN and AFTOX can also be used in much more detailed cases since they each ask for more details than provided by the problems in the workbook.

The numerical solutions found for the advection equation, all three advection-diffusion equations, and the steady-state equation show the effect of violating the stability condition or the condition that handles oscillations in the solution.

Each of these results show possibilities for further research in the area of air pollution modeling, some of which will be discussed in Chapter V.

V. Conclusion and Recommendations

5.1 Conclusion

The purpose of this study was to analyze models which use analytical solutions in their calculations of pollutant concentrations and to find numerical solutions of the various partial differential equations used in pollution modeling.

Many air pollution transport models use Gaussian analytical equations to solve the partial differential equations used in pollution modeling. This study has looked at three such models and made comparisons between them using example problems which deal with finding pollutant concentrations at a given location and time and finding the maximum concentration and its location.

The Gaussian plume model used in many air pollution transport models has difficulties when the wind speeds are low to calm. This research has looked at numerically solving the advection equation, advection-diffusion equation, and the steady-state equation using low wind speeds in the calculations. The results of these calculations were compared to exact solutions.

5.2 Recommendations

Pollution modeling has many areas of interest to consider for further research based on this research. The first is to look at more of the models which use analytical equations in their calculations. This research looked at SCREEN, AFTOX, and GAUSPLUM. Two other models to consider are TOXST (12), and TOXLT (11) which are referenced in the bibliography of this thesis. These models look at short term and long term exposure of pollutants.

Another area of possible further research could be continuing the two and three-dimensional research by looking at variable advection and diffusion coefficients. This research looked at those coefficients as constant in determining solutions. Although

the advection coefficient is often averaged as the mean wind speed, it can be a made variable function dependent on some standard deviation of the wind speed. The diffusion coefficient is in reality variable and dependent on the distance from the source. These variations could be added to numerical (finite- difference) schemes. Also, other numerical schemes, such as the finite element method, could be applied to the same equations evaluated in this research.

Another subject of possible further research could be expanding the regions of concern and using parallel computing systems in finding solutions to the equations to better simulate the boundaries in real life, like $x, y, z \rightarrow \pm\infty$. This research somewhat limited in range due to storage limits encountered using the Ada progamming language.

A fourth area of possible further research could be looking at the area of risk assessment of pollutants done by models currently available through the Environmental Protection Agency and other sources of pollution models.

Appendix A. GAUSPLUM Ada Code

This Appendix contains the Ada source code for the program GAUSPLUM. The description for the variables used in the programs is within each program in the form of comment blocks before each procedure within the program. Comment lines start with a double hyphen and comment blocks begin and end with a line of hyphens.

```
-- FILE:  gausplum.a
-- PROJECT:  Gaussian Plume Model
-- DATE:  16 APR 93
-- VERSION: Version 1.0
-- AUTHORS:  Capt Dave Paal
-- DESCRIPTION:  This program calculates the concentration of
--  pollutants in a plume using a Gaussian Plume Equation.  The
--  program asks the user for the source strength, the surface
--  wind speed, coordinates of the receptor(x,y,z), Pasquill
--  Gifford stability condition, and the effective height of the
--  plume.  The diffusion terms, sigmas, are calculated using the
--  Pasquill Gifford sigmas, and x given by the user.
-- OPERATING SYSTEM:  UNIX/Sun Sparc Station
-- LANGUAGE:  Meridian Ada
-- FILES USED:
-------------------------------------------------------------------
--
-- CONTEXT CLAUSES
--
with text_io;
with Math_Lib;
use Math_Lib;
with my_integer_io;
with my_float_io;

procedure gausmod is
--
```

```
-- TYPE DECLARATIONS
--
-- type Stable_Type is (A,B,C,D,E,F);
--
--
-- GLOBAL VARIABLES AND EXCEPTIONS
   Q      : float;   --Source Strength, grams/second
   X      : float;   --Distance in X-direction in meters
   U      : float;   --Surface wind speed in meters/second
   Y      : float;   --Distance in Y-direction in meters
   Z      : float;   --Distance in Z-direction in meters
   H      : float;   --Effective height of plume in meters
   Stable  : character;  --Stability condition A to F
   SigmaY  : float;    --lateral diffusion term
   SigmaZ  : float;    --vertical diffusion term
   Concentration : float;  --calculated concentration
-------------------------------------------------------------------
--
-- PROCEDURE: Get_Info
-- DESCRIPTION:  This procedure asks the user for the following:
--  Source strength, coordinates of receptor (X,Y,Z), surface
--  wind speed, stability condition, and the effective height of
--  the plume.
--
-- INPUT PARAMETERS: Q : source strenght in grams per second,
--                   X : distance in x-direction in meters,
--                   Y : distance in y-direction in meters,
--                   Z : distanct in z-direction in meters,
--                   U : surface wind speed in meters per second,
--                   H : effective height of plume in meters,
--              Stable : stability condition, character A to F.
-- OUTPUT PARAMETERS:  Same as Input Parameters.
-- LOCAL VARIABLES: None.
-- GLOBALS USED:  Same as parameters.
-- CALLED BY: main.
-- CALLS:  None.
-------------------------------------------------------------------
procedure Get_Info (Q      : in out float;
                    X      : in out float;
                    Y      : in out float;
                    Z      : in out float;
                    U      : in out float;
```

```
                    H      : in out float;
                    Stable : in out character) is

   begin
    Text_Io.put_line("Enter the Source Strength in grams per
second.");
    My_Float_Io.get(Q);
    Text_Io.put_line("Enter the distance in the X-direction in
meters.");
    My_Float_Io.get(X);
    Text_Io.put_line("Enter the Y-direction distance in
meters.");
    My_Float_Io.get(Y);
    Text_Io.put_line("Enter the Z-direction distance in
meters.");
    My_Float_Io.get(Z);
    Text_Io.put_line("Enter the surface wind speed in
meters/second.");
    My_Float_Io.get(U);
    Text_Io.put_line("Enter the effective height of the plume in
meters.");
    My_Float_Io.get(H);
    Text_Io.put_line("Enter the stability condition, in upper
case.");
    Text_Io.get(Stable);
end Get_Info;
-------------------------------------------------------------------
--
-- PROCEDURE:  Calc_PG_Sigma
-- DESCRIPTION:  This procedure calculates the Pasquill Gifford
--   diffusion terms sigma-y, and sigma-z for the given stability
--   condition, A to F, and the distance X in meters from the
--   source.
--   The general form of the equations are:
--       sigmay = (K1 * X)/[1.0 + (X/K2)]**K3,
--   and sigmaz = (K4 * X)/[1.0 + (X/K2)]**K5.
--   As found in Air Pollution Modeling by Paolo Zannetti on
--   p.149.  The constants K1 to K5 used in this procedure were
--   derived by Gifford in 1976 in a diffusion experiment in flat
--   terrain.
-- INPUT PARAMETERS: Stable : stability condition given by user,
--                        X : x-direction distance of receptor,
```

```
-- OUTPUT PARAMETERS: SigmaY : lateral diffusion term,
--                    SigmaZ : vertical diffusion term.
-- LOCAL VARIABLES: None.
-- GLOBALS USED:  Same as parameters.
-- CALLED BY: Main.
-- CALLS:  None.
-------------------------------------------------------------------
procedure Calc_PG_Sigma (Stable  : in character;
                         X     : in Float;
                               SigmaY  : in out Float;
                      SigmaZ  : in out Float) is

  begin
    case Stable is
      when 'A' =>
       SigmaY := (X * 0.250)/exp(0.189*ln(1.0 + (X/927.0)));
       SigmaZ := (X * 0.1020)/exp(-1.918*ln(1.0 + (X/927.0)));
      when 'B' =>
       SigmaY := (X * 0.202)/exp(0.162*ln(1.0 + (X/370.0)));
       SigmaZ := (X * 0.0962)/exp(-0.101*ln(1.0 + (X/370.0)));
      when 'C'=>
       SigmaY := (X * 0.134)/exp(0.134*ln(1.0 + (X/283.0)));
       SigmaZ := (X * 0.0722)/exp(0.102*ln(1.0 + (X/283.0)));
      when 'D' =>
       SigmaY := (X * 0.0787)/exp(0.135*ln(1.0 + (X/707.0)));
       SigmaZ := (X * 0.0475)/exp(0.465*ln(1.0 + (X/707.0)));
      when 'E' =>
       SigmaY := (X * 0.0566)/exp(0.137*ln(1.0 + (X/1070.0)));
       SigmaZ := (X * 0.0335)/exp(0.624*ln(1.0 + (X/1070.0)));
      when 'F' =>
       SigmaY := (X * 0.037)/exp(0.134*ln(1.0 + (X/1170.0)));
       SigmaZ := (X * 0.022)/exp(0.7*ln(1.0 + (X/1170.0)));
      when others =>
          Text_Io.put_line("Invalid stability condition");
   end case;
  end Calc_PG_Sigma;
-------------------------------------------------------------------
--
-- PROCEDURE:  Calc_Concentration
-- DESCRIPTION:  This procedure calculates the pollution
--    concentration for the user entered source strength,
--    distance, stability condition surface wind speed, and
```

```
--    effective height of the plume.
-- INPUT PARAMETERS: SigmaY : lateral diffusion term,
--                   SigmaZ : vertical diffusion term,
--                   Q      : source strength,
--                   H      : effective height of plume,
--                   Y      : distance in y-direction,
--                   Z      : distance in z-direction,
--                   U      : surface wind speed.
-- OUTPUT PARAMETERS:  Concentration : calculated concentration.

-- LOCAL VARIABLES: None.
-- GLOBALS USED:  Same as parameters.
-- CALLED BY: Main.
-- CALLS:  None.
-------------------------------------------------------------------
procedure Calc_Concentration (SigmaY : in Float;
                   SigmaZ : in Float;
                   Q      : in Float;
                   Y      : in Float;
                   Z      : in Float;
                   U      : in Float;
                   H      : in Float;
                   Concentration : out Float) is
    result1 : float;
    result2 : float;
    result3 : float;
    result4 : float;
    Pi      : float := 3.1415926;

  begin
    result1 := (Q/(2.0 * Pi * SigmaY * SigmaZ * U));
    result2 := exp((-1.0/2.0)*(Y/SigmaY)*(Y/SigmaY));
    result3 := exp((-1.0/2.0)*((Z-H)/SigmaZ)*((Z-H)/SigmaZ));
    result4 := exp((-1.0/2.0)*((Z+H)/SigmaZ)*((Z+H)/SigmaZ));
    Concentration := result1 * result2 * (result3 + result4);

  end Calc_Concentration;

-------------------------------------------------------------------
--
-- PROCEDURE: Print_Concentration
-- DESCRIPTION:  This procedure prints the output of the program.
```

```
--  It prints out the concentration and the coordinates of the
--  receptor, the source strength, effective height of the plume,
--  surface wind speed, and the stability condition given.
-- INPUT PARAMETERS:  Concentration : calculated concentration,
--                    X,Y,Z         : recptor coordinates,
--                    Q             : source strength,
--                    H             : effective height of plume,
--                    U             : surface wind speed,
--                    Stable        : stability condition.
-- OUTPUT PARAMETERS:  None.
-- LOCAL VARIABLES:  None.
-- GLOBALS USED:  All.
-- CALLED BY: Main.
-- CALLS:  None.
-----------------------------------------------------------------
procedure Print_Info (Concentration : in Float;
                      X, Y, Z       : in Float;
                      Q             : in Float;
                      U             : in Float;
                      H             : in Float;
                      Stable        : in character) is

   begin
     Text_Io.put_line("The following concentration results
from");
     Text_Io.put_line("these input variables:");
     Text_Io.put_line("Source strength =  ");
     My_Float_Io.put(Q);
     Text_Io.new_line;
     Text_Io.put_line("X distance =  ");
     My_Float_Io.put(X);
     Text_Io.new_line;
     Text_Io.put_line("Y distance =  ");
     My_Float_Io.put(Y);
     Text_Io.new_line;
     Text_Io.put_line("Z distance =  ");
     My_Float_Io.put(Z);
     Text_Io.new_line;
     Text_Io.put_line("Surface wind direction is  ");
     My_Float_Io.put(U);
     Text_Io.new_line;
     Text_Io.put_line("Effective height of the plume is  ");
```

```
    My_Float_Io.put(H);
    Text_Io.new_line;
    Text_Io.put_line("Stability condition given is ");
    Text_Io.put(Stable);
    Text_Io.new_line;
    Text_Io.put_line("The calculated concentration is  ");
    My_Float_Io.put(Concentration);
    Text_Io.new_line;

  end Print_Info;


-------------------------------------------------------------------
--
-- PROCEDURE: gausmod (a.k.a main)
-- DESCRIPTION: This is the main program and calls all the above
--              modules
-- INPUT PARAMETERS: none
-- OUTPUT PARAMETERS: none
-- LOCAL VARIABLES: none
-- GLOBALS USED: All.
-- CALLED BY:  none
-- CALLS: Get_Info, Calc_PG_Sigma, Calc_Concentration,
--         and Print_Info.
-------------------------------------------------------------------

begin --MAIN
  Get_Info(Q, X, Y, Z, U, H, Stable);
  Calc_PG_Sigma(Stable, X, SigmaY, SigmaZ);
  Calc_Concentration(SigmaY, SigmaZ, Q, Y, Z, U, H,
                     Concentration);
  Print_Info(Concentration, X, Y, Z, Q, U, H, Stable);
end gausmod;
```

Appendix B. Numerical solutions Ada code

The B-series appendices contain the Ada programming language programs which contain the finite difference schemes used to solve the advection equation, one-, two-, and three-dimensional advection-diffusion equations, and the two-dimensional steady- state equation.

Appendix B.1 has the program for the advection equation. Appendix B.2 has the program for the 1-D advection-diffusion equation. Appendix B.3 has the program for the 2-D advection-diffusion equation. Appendix B.4 has the program for the 3-D advection- diffusion equation. Appendix B.5 has the program for the 2-D steady-state equation.

The description for the variables used in the programs is within each program in the form of comment blocks before each procedure within the program. Comment lines start with a double hyphen and comment blocks begin and end with a line of hyphens.

B.1 Advection Equation Ada Code

```
-- FILE: ex131.a
-- PROJECT:  Thesis work
-- DATE: 15 Jul 93
-- VERSION: Version 1.0
-- AUTHORS:  Capt Dave Paal
-- DESCRIPTION:  This program solves a hyperbolic partial
--     differential equation using the Lax-Friedrichs scheme.
--     The partial differential equation to be solved is:
--
--          d/dt (U) + d/dx (U) = 0
--
-- OPERATING SYSTEM:  UNIX/Sun Sparc Station
-- LANGUAGE:  Meridian Ada
-- FILES USED: Output:  num131.dat.
-------------------------------------------------------------------
--
-- CONTEXT CLAUSES
--
with text_io;
with Math_Lib;
use Math_Lib;
with my_integer_io;
with my_float_io;

procedure prob131  is
--
-- TYPE DECLARATION
--
   type Vector is array (1..50) of float;
--
--
-- GLOBAL VARIABLES AND EXCEPTIONS
   V_Old        : Vector;
   V_New        : Vector;
   Delta_X      : float;
   Delta_T      : float;
   Min_X        : float;
   Max_X        : float;
   T_Value      : float;
   T            : float;
   Outfile      : Text_Io.File_Type;
```

```
-------------------------------------------------------------
-- PROCEDURE:  Get information for calculation
-- DESCRIPTION:  This procedure gets the values of Delta_X,
--    Delta_T, Min_X, Max_X, and Iterations.
-- INPUT PARAMETERS: None.
-- OUTPUT PARAMETERS:  Del_X : incrementor for space
--                     Del_T : incrementor for time
--                     Low_X : lower bound for X
--                     Hi_X  : upper bound for X
--                     Time  : time value wanted for calculation
-- LOCAL VARIABLES:  None.
-- GLOBALS USED: None.
-- CALLED BY: main.
-- CALLS:   none.
-------------------------------------------------------------
procedure Get_Info (Del_X : in out float;
                    Del_T : in out float;
                    Low_X   : in out float;
                    Hi_X    : in out float;
                    Time : in out float) is

begin

  Text_Io.put_line("Enter the value (Floating point, ie 0.1) of
Delta X.");
  My_Float_Io.get(Del_X);
  Text_Io.put_line("Enter the value (Floating point, ie 0.1) of
Delta T.");
  My_Float_Io.get(Del_T);
  Text_Io.put_line("Enter the minimum value (Floating point, ie
-2.0) of X.");
  My_Float_Io.get(Low_X);
  Text_Io.put_line("Enter the maximum value (Floating point, ie
3.0) of X.");
  My_Float_Io.get(Hi_X);
  Text_Io.put_line("Enter calculation time value (Floating point,
ie 1.6).");
  My_Float_Io.get(Time);
end Get_Info;
-------------------------------------------------------------
-- PROCEDURE:  Initial Vector
-- DESCRIPTION:  This procedure initializes the vector
```

```
--    using Del_X, Low_X, Hi_X.  If |x|<=1.0 the vector element
--    is given the value 1-|x|, otherwise the vector element is
-     zero.
-- INPUT PARAMETERS: V_Old : vector to represent x direction
--                   Del_X : space incrementer
--                   Del_T : time incrementer
--                   Low_X : lower bound for X
--                   Hi_X   : upper bound for X
-- OUTPUT PARAMETERS:  V_Old : initial vector for problem
-- LOCAL VARIABLES:  X : Value of X for each iteration of loop
--                   Count : number of X increments.
-- GLOBALS USED: None.
-- CALLED BY: main.
-- CALLS:   none.
-----------------------------------------------------------------
procedure Initial_Vector (V_Old      : in out Vector;
                Del_X, Low_X, Hi_X  : in  float) is

    X : float := Low_X;
    count : integer := integer((Hi_X - Low_X)/Del_X);

  begin
      for i in 1..count loop
         if abs(X) <= 1.0 then
         V_Old(i) := 1.0 - abs(X);
         else
            V_Old(i) := 0.0;
         end if;
         X := X + Del_X;
      end loop;
   end Initial_Vector;
-----------------------------------------------------------------
--
-- PROCEDURE: Compute New Vector
-- DESCRIPTION:  This procedure calculates the solution vector
--        in increments each time it is called by the main
--        program.
-- INPUT PARAMETERS:  V_Old : Vector for calculating answer
--                    V_New : Vector for calculating answer
--                    Del_X : incrementor for space
--                    Del_T : incrementor for time
--                    Low_X   : lower bound for X
```

```
--                    Hi_X    : upper bound for X
-- OUTPUT PARAMETERS: V_Old : Becomes vector with values for this
--                           iteration
--                    V_New : Vector for calculating answer
-- LOCAL VARIABLES:  Lambda : float Delta_T/Delta_X
--                Count : Counter for Vector array
-- GLOBALS USED:  None.
-- CALLED BY:  main.
-- CALLS:   none.
-------------------------------------------------------------------
procedure Compute_New(V_Old : in out Vector;
            V_New : in out Vector;
            Del_X, Del_T, Low_X, Hi_X  : in float) is

      Lambda : float := Del_T/Del_X;
      Count  : integer := integer((Hi_X - Low_X)/Del_X);

  begin
    V_New(1) :=0.0;
    for i in 2..(Count - 1) loop
      V_New(i) := 0.5*(V_Old(I+1) + V_Old(I-1)) -
                  0.5*Lambda*(V_Old(I+1) - V_Old(I-1));
    end loop;
    V_New(Count) := V_New(Count-1);
    for i in 1..Count loop
      V_Old(i) := V_New(i);
    end loop;
end Compute_New;
-------------------------------------------------------------------
--
-- PROCEDURE: Print_Info
-- DESCRIPTION:  This procedure prints the output of the program.

--  It prints out the vector with the points for the time
--  iteration given.
-- INPUT PARAMETERS:  V_Old : Vector for calculating answer
--                    V_New : Vector for calculating answer
--                    Del_X : incrementor for space
--                    Del_T : incrementor for time
--                    Low_X : lower bound for X
--                    Hi_X  : upper bound for X
-- OUTPUT PARAMETERS: V_New : Vector answer
```

```
-- LOCAL VARIABLES:  Count   : integer counter for array
-- GLOBALS USED: None.
-- CALLED BY: main.
-- CALLS:  none.
-----------------------------------------------------------------
procedure Print_Info(V_New : in out Vector;
             Del_X, Del_T, Low_X, Hi_X, Time  : in float;
                   File : in out Text_Io.File_Type) is

      Count  : integer := integer((Hi_X - Low_X)/Del_X);
      X_Value : float := Low_X+Del_X;
   begin
     Text_Io.put_line("The following are the results using the
Lax- ");
     Text_Io.put_line("Friedrichs scheme on a hyperbolic partial
difeq.");
     Text_Io.put_line("The resulting vector is ");
     Text_Io.new_line(File);
     Text_Io.put_line(File,"# The following are the results using
the ");
     Text_Io.put_line(File,"# Lax-Friedrichs scheme on a
hyperbolic");
     Text_Io.put_line(File,"# partial difeq. The resulting vector
is ");
     Text_Io.put(File,"# Output for Min_x=");
     My_Float_Io.put(File,Low_X,3,2,0);
     Text_Io.new_line(File);
     Text_Io.put(File,"# Max_x=");
     My_Float_Io.put(File,Hi_X,3,2,0);
     Text_Io.new_line(File);
     Text_Io.put(File,"# Delta_X=");
     My_Float_Io.put(File,Del_X,3,5,0);
     Text_Io.new_line(File);
     Text_Io.put(File,"# Delta_T=");
     My_Float_Io.put(File,Del_T,3,5,0);
     Text_Io.new_line(File);
     Text_Io.put(File,"# Time of calculation=");
     My_Float_Io.put(File,Time,3,5,0);
     Text_Io.new_line(File);
     for i in 1..Count loop
         My_Float_Io.put(X_Value,1,2,0);
         Text_Io.put("     ");
```

```
          My_Float_Io.put(V_New(i),1,6,2);
          Text_Io.new_line;
          Text_Io.set_col(File,1);
          My_Float_Io.put(File,X_Value,1,2,0);
          Text_Io.set_col(File,10);
          My_Float_Io.put(File,V_New(i),1,6,2);
          X_Value := X_Value + Del_X;
      end loop;
      Text_Io.new_line;
      Text_Io.new_line;
   end Print_Info;
-------------------------------------------------------------------
-- PROCEDURE:  Main
-- DESCRIPTION: This is the main program and calls all the above
--              modules
-- INPUT PARAMETERS: none
-- OUTPUT PARAMETERS: none
-- LOCAL VARIABLES: none
-- GLOBALS USED: All.
-- CALLED BY:  none
-- CALLS:  Get_Info, Initial_Vector, Compute_New, Print_Info.
-------------------------------------------------------------------
begin --MAIN
   Text_Io.Create(Outfile,Text_Io.Out_File,"num131.dat");
   Get_Info (Delta_X, Delta_T, Min_X, Max_X, T_Value);
   Initial_Vector (V_Old, Delta_X, Min_X, Max_X);
   T := Delta_T;
   while T <= T_Value loop
     Compute_New(V_Old, V_New, Delta_X, Delta_T, Min_X, Max_X);
     T := T + Delta_T;
   end loop;
   Print_Info(V_New, Delta_X, Delta_T, Min_X, Max_X, T_Value,
            Outfile);
   Text_Io.Close(Outfile);
end prob131;
```

B.2 1-D Advection-Diffusion Equation Ada Code

```
-- FILE: exadvdif.a
-- PROJECT:  Thesis work
-- DATE: 19 Jul 93
-- VERSION: Version 1.0
-- AUTHORS:  Capt Dave Paal
-- DESCRIPTION:  This program solves a hyperbolic partial
--     differential equation using the forward-time and
--     central-space forward difference scheme.
--     The partial differential equation to be solved it:
--
--         d/dt (U) + d/dx (U) =  b * d**2/dx**2 (U)
--
-- OPERATING SYSTEM:  UNIX/Sun Sparc Station
-- LANGUAGE:  Meridian Ada
-- FILES USED: Output: advdif1.dat.
-----------------------------------------------------------------
--
-- CONTEXT CLAUSES
--
with text_io;
with Math_Lib;
use Math_Lib;
with my_integer_io;
with my_float_io;

procedure exadv1  is
--
-- TYPE DECLARATION
--
   type Vector is array (1..50) of float;
--
--
-- GLOBAL VARIABLES AND EXCEPTIONS
   V_Old        : Vector;
   V_New        : Vector;
   Delta_X      : float;
   Delta_T      : float;
   Min_X        : float;
   Max_X        : float;
   T_Value      : float;
   A, B         : float;
```

```
   Bmu          : float;
   T            : float;
   Outfile      : Text_Io.File_Type;
-------------------------------------------------------------------
-- PROCEDURE:  Get information for calculation
-- DESCRIPTION:  This procedure gets the values of Del_X, Del_T,
--    Low_X, Hi_X, and Time.
-- INPUT PARAMETERS: None.
-- OUTPUT PARAMETERS:  Del_T : time incrementor
--                     Del_X : space incrementor
--                     Low_X : lower bound for X
--                     Hi_X  : upper bound for X
--                     A, B  : constants (positive)
--                     Time  : time calculation is wanted
-- LOCAL VARIABLES:  None.
-- GLOBALS USED:
-- CALLED BY: main.
-- CALLS:   none.
-------------------------------------------------------------------
procedure Get_Info (Del_X : in out float;
              Del_T : in out float;
                    Low_X   : in out float;
                    Hi_X    : in out float;
                    A, B    : in out float;
                 Time : in out float) is

begin

  Text_Io.put_line("Enter the value (Floating point, ie 0.1) of
Delta X.");
  My_Float_Io.get(Del_X);
  Text_Io.put_line("Enter the value (Floating point, ie 0.05) of
Delta T.");
  My_Float_Io.get(Del_T);
  Text_Io.put_line("Enter the minimum value (Floating point, ie
-2.0) of X.");
  My_Float_Io.get(Low_X);
  Text_Io.put_line("Enter the maximum value (Floating point, ie
3.0) of X.");
  My_Float_Io.get(Hi_X);
  Text_Io.put_line("Enter the value (Positive Floating point, ie
0.5) of A.");
```

```
  My_Float_Io.get(A);
  Text_Io.put_line("Enter the value (Positive Floating point, ie
0.5) of B.");
  My_Float_Io.get(B);
  Text_Io.put_line("Enter calculation time value (Floating point,
ie 1.6).");
  My_Float_Io.get(Time);
  end Get_Info;
-----------------------------------------------------------------
-- PROCEDURE:  Initial Vector
-- DESCRIPTION:  This procedure initializes the vector
--    using Del_X, Low_X, Hi_X. If |x|<=1.0 the vector element
--    is given the value 1-|x|, otherwise the vector element is
--    zero.
-- INPUT PARAMETERS:  Del_X : space incrementor
--                    Low_X, Hi_X : min and max range
--                    A, B : positive constants
-- OUTPUT PARAMETERS:  V_Old : V(0)
-- LOCAL VARIABLES:  X : float
--                   Count : number of X increments
--                   Mu : Del_T/(Del_X * Del_X)
-- GLOBALS USED: none.
-- CALLED BY: main.
-- CALLS:   none.
-----------------------------------------------------------------
procedure Initial_Vector (V_Old      : in out Vector;
                Del_X, Del_T, Low_X, Hi_X, A, B  : in  float) is

    X : float := Low_X;
    count : integer := integer((Hi_X - Low_X)/Del_X);
    Mu : float := Del_T/(Del_X * Del_X);

  begin
      if B*Mu > 0.5 then
         Text_Io.put_line("B*Mu > 0.5 causes unstable results");
      else
         for i in 1..count loop
            if abs(X) <= 1.0 then
               V_Old(i) := 1.0 - abs(X);
            else
               V_Old(i) := 0.0;
            end if;
```

```
            X := X + Del_X;
          end loop;
        end if;
    end Initial_Vector;
-------------------------------------------------------------
--
-- PROCEDURE: Compute Vector answer for time wanted.
-- DESCRIPTION:  This procedure calculates new vector
-- INPUT PARAMETERS:  V_Old : Vector for V(N)
--                    V_New : Vector for V(N+1)
--                    Del_X : incrementor for space
--                    Del_T : incrementor for time
--                    Low_X : lower bound for X
--                    Hi_X  : upper bound for X
--                    A, B  : positive constants
-- OUTPUT PARAMETERS: V_Old : Vector for V(N)
--                    V_New : Vector for V(N+1)
-- LOCAL VARIABLES:  Mu : float Del_T/Del_X**2
--                Count : Counter for Vector array
--                Alpha : (Del_X * A)/(2*B)
-- GLOBALS USED: None.
-- CALLED BY:  main.
-- CALLS:   none.
-------------------------------------------------------------
procedure Compute_New(V_Old : in out Vector;
             V_New : in out Vector;
             Del_X, Del_T, Low_X, Hi_X, A, B  : in float) is

      Count  : integer := integer((Hi_X - Low_X)/Del_X);
      Mu     : float := Del_T/(Del_x * Del_X);
      Alpha  : float := (Del_X * A)/(2.0*B);

  begin
    V_New(1) :=0.0;
    for i in 2..(Count - 1) loop
      V_New(i) := ((1.0 - 2.0 * B * Mu) * V_Old(i)) + (B * Mu *

                (1.0 - Alpha) * V_Old(i+1)) +
                  (B * Mu * (1.0 + Alpha) * V_Old(i-1));
    end loop;
    V_New(Count) := V_New(Count-1);
    for i in 1..Count loop
```

```
      V_Old(i) := V_New(i);
    end loop;
end Compute_New;
-------------------------------------------------------------------
-- PROCEDURE: Print_Info
-- DESCRIPTION:  This procedure prints the output of the program.
--  It prints the point and corresponding concentration for the
--  wanted time to both the screen and an output file named
--  advdif1.dat.
-- INPUT PARAMETERS:  V_New : Vector answer
--                    Del_X : incrementor for space
--                    Del_T : incrementor for time
--                    Low_X : lower bound for X
--                    Hi_X  : upper bound for X
-- OUTPUT PARAMETERS: V_New : Vector answer
-- LOCAL VARIABLES:  Count  : integer counter for array
--                  X_Value : value of x for each iteration
--                                          through vector
--                    Alpha : (Del_X * A) / (2.0 * B) <= 1.0 for
--                                          no oscillation
--                    Bmu   : (B * Del_T)/(Del_X * Del_X) <= 0.5
--                                          for stability
-- GLOBALS USED: None.
-- CALLED BY: main.
-- CALLS:  none.
-------------------------------------------------------------------
procedure Print_Info(V_New : in out Vector;
            Del_X, Del_T, Low_X, Hi_X, Time, A, B  : in float;
                  File : in out Text_Io.File_Type) is

      Count  : integer := integer((Hi_X - Low_X)/Del_X);
      X_Value : float := Low_X+Del_X;
      Alpha : float := (Del_X * A) / (2.0 * B);
      Bmu   : float := (B * Del_T)/(Del_X * Del_X);
begin
   Text_Io.put_line("The following are the results using forward

                    time ");
   Text_Io.put_line("central space scheme on a hyperbolic partial

                     difeq.");
   Text_Io.put_line("The resulting vector is ");
```

```
      Text_Io.put_line(File,"# The following are the results using

                              the ");
      Text_Io.put_line(File,"# forward time/central space scheme on

                                a ");
      Text_Io.put_line(File,"# Ct + aCx = bCxx.");
      Text_Io.put_line(File,"# The resulting vector is: ");
      Text_Io.put(File,"# Output for Min_x=");
      My_Float_Io.put(File,Low_X,3,2,0);
      Text_Io.new_line(File);
      Text_Io.put(File,"# Max_x=");
      My_Float_Io.put(File,Hi_X,3,2,0);
      Text_Io.new_line(File);
      Text_Io.put(File,"# Delta X=");
      My_Float_Io.put(File,Del_X,3,5,0);
      Text_Io.new_line(File);
      Text_Io.put(File,"# Delta_T=");
      My_Float_Io.put(File,Del_T,3,5,0);
      Text_Io.new_line(File);
      Text_Io.put(File,"# Time of calculation=");
      My_Float_Io.put(File,Time,3,5,0);
      Text_Io.new_line(File);
      Text_Io.put(File,"# Alpha= (Delta_X * A)/2B=");
      My_Float_Io.put(File,Alpha,3,5,0);
      Text_Io.new_line(File);
      Text_Io.put(File,"# Bmu= B*Delta_T/Delta_X**2=");
      My_Float_Io.put(File,Bmu,3,5,0);
      Text_Io.new_line(File);
      for i in 1..Count loop
         Text_Io.set_col(File,1);
         My_Float_Io.put(File,X_Value,2,2,0);
         Text_Io.set_col(File,15);
         My_Float_Io.put(File,V_New(i),2,6,2);
         X_Value := X_Value + Del_X;
      end loop;
      Text_Io.new_line;
      Text_Io.new_line;
   end Print_Info;
   ------------------------------------------------------------------
   -- PROCEDURE:  Example1.3.1 (a.k.a main)
   -- DESCRIPTION: This is the main program and calls all the above
```

```
--              modules
-- INPUT PARAMETERS: none
-- OUTPUT PARAMETERS: none
-- LOCAL VARIABLES: none
-- GLOBALS USED: All.
-- CALLED BY:  none
-- CALLS:  Get_Info, Initial_Vector, Compute_New, Print_Info.
-------------------------------------------------------------------
begin --MAIN
   Text_Io.Create(Outfile,Text_Io.Out_File,"advdif1.dat");
   Get_Info (Delta_X, Delta_T, Min_X, Max_X, A, B, T_Value);
   Initial_Vector (V_Old, Delta_X, Delta_T, Min_X, Max_X, A, B);
   Bmu := B*Delta_T/(Delta_X*Delta_X);
   if Bmu > 0.5 then
      Text_Io.put_line("Unstable result, try again.");
   else
      T := Delta_T;
      while T <= T_Value loop
        Compute_New(V_Old, V_New, Delta_X, Delta_T, Min_X, Max_X,

                    A, B);
        T := T + Delta_T;
      end loop;
      Print_Info(V_New, Delta_X, Delta_T, Min_X, Max_X, T_Value,

                 A, B, Outfile);
   end if;
   Text_Io.Close(Outfile);
end exadv1;
```

B.3 2-D Advection-Diffusion Equation Ada Code

```
-- FILE: exadvdif2.a
-- PROJECT:  Thesis work, example calculation
-- DATE: 23 Jul 93
-- VERSION: Version 1.0
-- AUTHORS:  Capt Dave Paal
-- DESCRIPTION:  This program solves a partial differential
--     equation using the forward time and central space.
--     The partial differential equation to be solved is:
--
--         Ct + A*Cx = K1*Cxx + K2*Cyy
--
-- OPERATING SYSTEM:  UNIX/Sun Sparc Station
-- LANGUAGE:  Meridian Ada
-- FILES USED: Output: expltest.dat
-------------------------------------------------------------------
--
-- CONTEXT CLAUSES
--
with text_io;
with Math_Lib;
use Math_Lib;
with my_integer_io;
with my_float_io;

procedure ctprob  is
--
-- TYPE DECLARATION
--
   type Matrix is array (1..80,1..80) of float;
--
--
-- GLOBAL VARIABLES AND EXCEPTIONS
   C_N          : Matrix;
   C_New        : Matrix;
   Delta_X      : float;
   Delta_Y      : float;
   Delta_T      : float;
   Min_X        : float;
   Max_X        : float;
   Min_Y        : float;
   Max_Y        : float;
```

```
   A            : float;
   K1           : float;
   K2           : float;
   T_Value      : float;
   T            : float;
   Outfile      : Text_Io.File_Type;
-------------------------------------------------------------------
-- PROCEDURE:  Get information for calculation
-- DESCRIPTION:  This procedure gets the values of Del_X, Del_T,
--    Del_Y, Low_X, Hi_X, Low_Y, Hi_Y, A, K1, K2, and Time.
-- INPUT PARAMETERS: None.
-- OUTPUT PARAMETERS:  Del_X : incrementor for space
--                     Del_T : incrementor for time
--                     Del_Y : incrementor for Y direction
--                     Low_X   : lower bound for X
--                     Hi_X   : upper bound for X
--                     Low_Y   : lower bound for Y
--                     Hi_Y   : upper bound for Y
--                     A       : constant wind speed
--                     K1, K2  : diffusivity constants
--                     Time : wanted time level
-- LOCAL VARIABLES:  None.
-- GLOBALS USED: Del_X, Del_T, Del_Y, Low_X, Hi_X, Low_Y, Hi_Y,
--               A, K1, K2, and Time.
-- CALLED BY: main.
-- CALLS:   none.
-------------------------------------------------------------------
procedure Get_Info (Del_X : in out float;
                    Del_Y : in out float;
                    Del_T : in out float;
                    Low_X : in out float;
                    Hi_X  : in out float;
                    Low_Y : in out float;
                    Hi_Y  : in out float;
                    A     : in out float;
                    K1, K2: in out float;
                    Time  : in out float) is

   begin

     Text_Io.put_line("Enter the value (Floating point) of Delta
X. ");
```

```
     My_Float_Io.get(Del_X);
     Text_Io.put_line("Enter the value (Floating point) of Delta
Y. ");
     My_Float_Io.get(Del_Y);
     Text_Io.put_line("Enter the value (Floating point) of Delta
T. ");
     My_Float_Io.get(Del_T);
     Text_Io.put_line("Enter the minimum value (Floating point)
of X. ");
     My_Float_Io.get(Low_X);
     Text_Io.put_line("Enter the maximum value (Floating point)
of X. ");
     My_Float_Io.get(Hi_X);
     Text_Io.put_line("Enter the minimum value (Floating point)
of Y. ");
     My_Float_Io.get(Low_Y);
     Text_Io.put_line("Enter the maximum value (Floating point)
of Y. ");
     My_Float_Io.get(Hi_Y);
     Text_Io.put_line("Enter the value (Positive Floating point)
of A. ");
     My_Float_Io.get(A);
     Text_Io.put_line("Enter the value (Positive Floating point)
of K1. ");
     My_Float_Io.get(K1);
     Text_Io.put_line("Enter the value (Positive Floating point)
of K2. ");
     My_Float_Io.get(K2);
     Text_Io.put_line("Enter the number of time value wanted
(float).");
     My_Float_Io.get(Time);
  end Get_Info;
-----------------------------------------------------------------
-- PROCEDURE:  Initial Matrix
-- DESCRIPTION:  This procedure initializes the matrix
--    using Del_X, Del_Y, Low_X, Hi_X, Low_Y, Hi_Y.
-- INPUT PARAMETERS:  C_N, Del_X, Del_Y, Low_X, Hi_X, Low_Y, Hi_Y
-- OUTPUT PARAMETERS:  C_N.
-- LOCAL VARIABLES:  X : X value incrementer.
--                   Y : Y value incrementer.
--                   Count_X : number of X increments.
--                   Count_Y : number of Y increments.
```

```
-- GLOBALS USED: C_N, Del_X, Del_Y, Low_X, Hi_X, Low_Y, Hi_Y.
-- CALLED BY: main.
-- CALLS:   none.
-----------------------------------------------------------------
procedure Initial_Matrix (C_N         : in out Matrix;
                Del_X, Del_Y : in float;
                Low_X, Hi_X, Low_Y, Hi_Y  : in  float) is

    X : float := Low_X;
    Y : float := Low_Y;
    Count_X : integer := integer((Hi_X - Low_X)/Del_X);
    Count_Y : integer := integer((Hi_Y - Low_Y)/Del_Y);
    R : float := sqrt(X*X + Y*Y);

  begin
      for j in 1..Count_Y loop
         X := Low_X;
         for i in 1..Count_X loop
            if (R >= 0.0 and R <= 1.0) then
            C_N(i,j) := 1.0 - R;
            else
               C_N(i,j) := 0.0;
            end if;
            X := X + Del_X;
            R := sqrt(X*X + Y*Y);
         end loop;
         Y := Y + Del_Y;
      end loop;
   end Initial_Matrix;
-----------------------------------------------------------------
--
-- PROCEDURE: Compute final Matrix
-- DESCRIPTION:  This procedure calculates new vector
-- INPUT PARAMETERS:  C_N : Matrix for C_N(i,j)
--                    C_New : Matrix for calculating answer
--                    Del_X : incrementor for space
--                    Del_T : incrementor for time
--                    Del_Y : incrementor for Y direction
--                    Low_X   : lower bound for X
--                    Hi_X   : upper bound for X
--                    Low_Y   : lower bound for Y
--                    Hi_Y   : upper bound for Y
```

```
--                      A       : constant
--                      K1      : X diffusivity constant
--                      K2      : Y diffusivity constant
-- OUTPUT PARAMETERS: C_N : Matrix for C(N)
--                      C_New : Matrix for calculating answer
-- LOCAL VARIABLES:  Mu : (A*Del_T)/(2.0*Del_X)
--                   Alpha : (K1*Del_T)/(Del_X*Del_X)
--                   Beta : (K2*Del_T)/(Del_Y*Del_Y)
--                   Count_Y : integer((Hi_Y -Low_Y)/Del_Y)
--                   Count_X : integer((Hi_X - Low_X)/Del_X)
-- GLOBALS USED: C_N, C_New, Del_X, Del_T, Del_Y, Low_X, Hi_X
--               Low_Y, Hi_Y, A, K1, K2.
-- CALLED BY:  main.
-- CALLS:   none.
-- ANALYSIS: O(1).
-----------------------------------------------------------------
procedure Compute_Final(C_N : in out Matrix;
                        C_New : in out Matrix;
                        Del_X, Del_T, Del_Y : in float;
                        Low_X, Hi_X, Low_Y, Hi_Y : in float;
                        A, K1, K2  : in float) is

     Mu : float := (A*Del_T)/(2.0*Del_X);
     Alpha : float := (K1*Del_T)/(Del_X*Del_X);
     Beta : float := (K2*Del_T)/(Del_Y*Del_Y);
     Count_X  : integer := integer((Hi_X - Low_X)/Del_X);
     Count_Y  : integer := integer((Hi_Y - Low_Y)/Del_Y);

  begin
    for i in 1..Count_X loop
       C_New(i,Count_Y) :=0.0;
       C_New(i,1) :=0.0;
    end loop;
    for i in 1..Count_Y loop
       C_New(Count_X,i) :=0.0;
       C_New(1,i) :=0.0;
    end loop;
    for j in 2..(Count_Y - 1) loop
       for i in 2..(Count_X - 1) loop
          C_New(i,j) := C_N(i,j) - Mu*(C_N(i+1,j) - C_N(i-1,j)) +
                        Alpha*(C_N(i+1,j) - 2.0*C_N(i,j) +
                             C_N(i-1,j)) +
```

```
                        Beta*(C_N(i,j+1) - 2.0*C_N(i,j) +
                            C_N(i,j-1));
        end loop;
      end loop;
      for i in 1..Count_X loop
        for j in 1..Count_Y loop
          C_N(i,j) := C_New(i,j);
        end loop;
      end loop;
end Compute_Final;
-------------------------------------------------------------
--
-- PROCEDURE: Print_Info
-- DESCRIPTION:  This procedure prints the output of the program.
--  It prints out the matrix with the points for the time
--  iteration given.
-- INPUT PARAMETERS:  C_New : Matrix for calculating answer
--                    Del_X : incrementor for space
--                    Del_T : incrementor for time
--                    Del_Y : incrementor for Y direction
--                    Low_X   : lower bound for X
--                    Hi_X   : upper bound for X
--                    Low_Y   : lower bound for Y
--                    Hi_Y   : upper bound for Y
-- OUTPUT PARAMETERS: C_New : Matrix answer
--                    File : output file
-- LOCAL VARIABLES:  Count_X, Count_Y : matrix integer counters
--                   X_Value, Y_Value : X and Y points
-- GLOBALS USED: C_New, Del_X, Del_T, Del_Y, Low_X, Hi_X,
--               Low_Y, Hi_Y, File.
-- CALLED BY: main.
-- CALLS:  none.
-------------------------------------------------------------
procedure Print_Info(C_New : in out Matrix;
                     Del_X, Del_T, Del_Y : in float;
                     Low_X, Hi_X, Low_Y, Hi_Y  : in float;
                     File : in out Text_Io.File_Type) is

      Count_X  : integer := integer((Hi_X - Low_X)/Del_X);
      Count_Y  : integer := integer((Hi_Y - Low_Y)/Del_Y);
      X_Value : float := Low_X;
      Y_Value : float := Low_Y;
```

```
   begin
     Text_Io.put_line("#The following are the results using the
forward time ");
     Text_Io.put_line("# central space for the partial
diferential equation:");
     Text_Io.put_line("#        Ct + A*Cx = K1*Cxx + K2*Cyy ");
     Text_Io.put_line("# The resulting matrix is: ");
     Text_Io.new_line;
     Text_Io.new_line(File,2);
     Text_Io.put_line(File,"# Output for a=.5, k1=.5, k2=.5");
     Text_Io.put_line(File,"# delta_x=.2, delta_y=.2,
delta_t=.02,t=.06");
     for j in 1..Count_Y loop
         X_Value := Low_X;
         for i in 1..Count_X loop
            Text_Io.set_col(File,1);
            My_Float_Io.put(File,X_Value,1,2,0);
            Text_Io.set_col(File,10);
            My_Float_Io.put(File, Y_Value,1,2,0);
            Text_Io.set_col(File,20);
            My_Float_Io.put(File,C_New(i,j),1,6,2);
            Text_Io.new_line(File);
            X_Value := X_Value + Del_X;
         end loop;
         Y_Value := Y_Value + Del_Y;
     end loop;
     Text_Io.new_line;
     Text_Io.new_line;
  end Print_Info;
-------------------------------------------------------------------
-- PROCEDURE:  Main
-- DESCRIPTION: This is the main program and calls all the above
--              modules
-- INPUT PARAMETERS: none
-- OUTPUT PARAMETERS: none
-- LOCAL VARIABLES: none
-- GLOBALS USED: All.
-- CALLED BY:  none
-- CALLS:  Get_Info, Initial_Matrix, Compute_Final, Print_Info.
-------------------------------------------------------------------
begin --MAIN
```

```
   Text_Io.Create(Outfile,Text_Io.Out_File,"expltest.dat");
   Get_Info (Delta_X, Delta_Y, Delta_T, Min_X, Max_X, Min_Y,
        Max_Y, A, K1, K2, T_Value);
   Initial_Matrix (C_N, Delta_X, Delta_Y, Min_X, Max_X, Min_Y,
                   Max_Y);
   T := Delta_T;
   while T <= T_Value loop
     Compute_Final(C_N, C_New, Delta_X, Delta_T, Delta_Y, Min_X,
                   Max_X, Min_Y, Max_Y, A, K1, K2);
     T := T + Delta_T;
   end loop;
   Print_Info(C_New, Delta_X, Delta_T, Delta_Y, Min_X, Max_X,
              Min_Y, Max_Y, Outfile);
   Text_Io.Close(Outfile);
end ctprob;
```

B.4 3-D Advection-Diffusion Equation Ada Code

```
-- FILE: threedprob.a
-- PROJECT:  Thesis work
-- DATE: 12 Aug 93
-- VERSION: Version 1.0
-- AUTHORS:  Capt Dave Paal
-- DESCRIPTION:  This program solves a partial differential
--     equation using the forward time and central space.
--     The partial differential equation to be solved is:
--
--         Ct + A*Cx = K1*Cxx + K2*Cyy + K3*Czz + F(X,Y,Z)
--
--    Where F(X,Y,Z) is the source term.
--    For this example the following conditions must hold for a
--    stable non-oscillating solution:
--                      (Ki*Del_T)/(Del_X*Del_X)<= 1/8
--            and        (A*Del_X)/(2*Ki)<= 1.0
--            where the i subscript is 1,2,or 3 for X,Y,or Z
--            diffusivity terms.
--   These conditions limit the usefulness of this program
--   because they more or less make the diffusivity terms
--   constant when in reality they are variable
--   (K(i)=sigma(i)**2/(2*time).
--   Another limitation of this program is that the source term
--   is hard coded as F(0,0,Stack_Height) and the program is
--   dependent on whether the Delta_X,(Y and Z) terms and the
--   minimum values of each will increment to X=0, Y=0,
--   and Z= height.
-- OPERATING SYSTEM:  UNIX/Sun Sparc Station
-- LANGUAGE:  Meridian Ada
-- FILES USED: Output: test3d.dat
-------------------------------------------------------------------
--
-- CONTEXT CLAUSES
--
with text_io;
with Math_Lib;
use Math_Lib;
with my_integer_io;
with my_float_io;

procedure threed  is
```

```
--
-- TYPE DECLARATION
--
   type Tri_Matrix is array (0..50,0..50,0..50) of float;
--
--
-- GLOBAL VARIABLES AND EXCEPTIONS
   C_N          : Tri_Matrix;
   C_New        : Tri_Matrix;
   C_Calc       : Tri_Matrix;
   Delta_X      : float;
   Delta_Y      : float;
   Delta_Z      : float;
   Delta_T      : float;
   Min_X        : float;
   Max_X        : float;
   Min_Y        : float;
   Max_Y        : float;
   Min_Z        : float;
   Max_Z        : float;
   A            : float;
   T_Value      : float;
   Q_Value      : float;
   Stk_Hgt      : float;
   T            : float;
   Outfile      : Text_Io.File_Type;
-------------------------------------------------------------------
-- PROCEDURE:  Get information for calculation
-- DESCRIPTION:  This procedure gets the values of Del_X, Del_T,
--    Del_Y, Del_Z, Low_X, Hi_X, Low_Y, Hi_Y, Low_Z, Hi_Z,
--    A, and Time.
--
-- INPUT PARAMETERS: None.
-- OUTPUT PARAMETERS:  Del_X : incrementor for space
--                     Del_T : incrementor for time
--                     Del_Y : incrementor for Y direction
--                     Del_Z : incrementor for Z direction
--                     Low_X   : lower bound for X
--                     Hi_X   : upper bound for X
--                     Low_Y   : lower bound for Y
--                     Hi_Y   : upper bound for Y
--                     Low_Z   : lower bound for Z
```

```
--                      Hi_Z   : upper bound for Z
--                      A      : constant wind speed
--                      Time : wanted time level
-- LOCAL VARIABLES:  None.
-- GLOBALS USED: None.
-- CALLED BY: main.
-- CALLS:   none.
-------------------------------------------------------------------
------------
procedure Get_Info (Del_X : in out float;
                    Del_Y : in out float;
                    Del_Z : in out float;
                    Del_T : in out float;
                    Low_X : in out float;
                    Hi_X  : in out float;
                    Low_Y : in out float;
                    Hi_Y  : in out float;
                    Low_Z : in out float;
                    Hi_Z  : in out float;
                    A     : in out float;
                    Time  : in out float;
                    Quantity : in out float;
                    Height : in out float) is

   begin

     Text_Io.put_line("Enter the value (Floating point) of Delta
X. ");
     My_Float_Io.get(Del_X);
     Text_Io.put_line("Enter the value (Floating point) of Delta
Y. ");
     My_Float_Io.get(Del_Y);
     Text_Io.put_line("Enter the value (Floating point) of Delta
Z. ");
     My_Float_Io.get(Del_Z);
     Text_Io.put_line("Enter the value (Floating point) of Delta
T. ");
     My_Float_Io.get(Del_T);
     Text_Io.put_line("Enter the minimum value (Floating point)
of X. ");
     My_Float_Io.get(Low_X);
     Text_Io.put_line("Enter the maximum value (Floating point)
```

```
of X. ");
     My_Float_Io.get(Hi_X);
     Text_Io.put_line("Enter the minimum value (Floating point)
of Y. ");
     My_Float_Io.get(Low_Y);
     Text_Io.put_line("Enter the maximum value (Floating point)
of Y. ");
     My_Float_Io.get(Hi_Y);
     Text_Io.put_line("Enter the minimum value (Floating point)
of Z. ");
     My_Float_Io.get(Low_Z);
     Text_Io.put_line("Enter the maximum value (Floating point)
of Z. ");
     My_Float_Io.get(Hi_Z);
     Text_Io.put_line("Enter the value (Positive Floating point)
of A. ");
     My_Float_Io.get(A);
     Text_Io.put_line("Enter the number of time value wanted
(float).");
     My_Float_Io.get(Time);
     Text_Io.put_line("Enter the value (Positive Floating point)
of Q. ");
     My_Float_Io.get(Quantity);
     Text_Io.put_line("Enter the height of the stack (float).");
     My_Float_Io.get(Height);
  end Get_Info;
-------------------------------------------------------------------
--
-- PROCEDURE: Compute final Tri_Matrix
-- DESCRIPTION:  This procedure calculates final answer through
--    iterations using a finite difference scheme with forward
--    time/central space.
-- INPUT PARAMETERS:  C_N : Tri_Matrix for C_N(i,j)
--                    C_New : Tri_Matrix for calculating answer
--                    Del_X : incrementor for space
--                    Del_T : incrementor for time
--                    Del_Y : incrementor for Y direction
--                    Del_Z : incrementor for Z direction
--                    Low_X : lower bound for X
--                    Hi_X  : upper bound for X
--                    Low_Y : lower bound for Y
--                    Hi_Y  : upper bound for Y
```

```
--                    Low_Z : lower bound for Z
--                    Hi_Z  : upper bound for Z
--                    A     : constant
--                    Time  : counter for time iteration
-- OUTPUT PARAMETERS: C_N   : Tri_Matrix for C(N)
--                    C_New : Tri_Matrix for calculating answer
-- LOCAL VARIABLES: Mu      : (A*Del_T)/(2.0*Del_X)
--                  K1      : X diffusivity constant
--                  K2      : Y diffusivity constant
--                  K3      : Z diffusivity constant
--                  Alpha   : (K1*Del_T)/(Del_X*Del_X)
--                  Beta    : (K2*Del_T)/(Del_Y*Del_Y)
--                  Gamma   : (K3*Del_T)/(Del_Z*Del_Z)
--                  Count_Y : integer((Hi_Y -Low_Y)/Del_Y)
--                  Count_X : integer((Hi_X - Low_X)/Del_X)
--                  Count_Z : integer((Hi_Z - Low_Z)/Del_Z)
--                  X       : counter for X axis
--                  Y       : counter for Y axis
--                  Z       : counter for Z axis
--                  Fofxyz  : source forcing function
--                  Pi      : the number Pi
-- GLOBALS USED: None.
-- CALLED BY:  main.
-- CALLS:   none.
-------------------------------------------------------------------
procedure Compute_Final(C_N : in out Tri_Matrix;
                  C_New : in out Tri_Matrix;
                  C_Calc : in out Tri_Matrix;
                  T, Del_X, Del_Y, Del_Z, Del_T : in float;
                  Low_X, Hi_X, Low_Y : in float;
                  Hi_Y, Low_Z, Hi_Z : in float;
                  A, Quantity, Height  : in float) is


      K1 : float := 1.0;
      K2 : float := 1.0;
      K3 : float := 1.0;
      Mu : float := (A*Del_T)/(2.0*Del_X);
      Alpha : float := (K1*Del_T)/(Del_X*Del_X);
      Beta : float := (K2*Del_T)/(Del_Y*Del_Y);
      Gamma : float := (K3*Del_T)/(Del_Z*Del_Z);
      Count_X  : integer := integer((Hi_X - Low_X)/Del_X);
```

```
    Count_Y  : integer := integer((Hi_Y - Low_Y)/Del_Y);
    Count_Z  : integer := integer((Hi_Z - Low_Z)/Del_Z);
    X : float := Low_X;
    Y : float := Low_Y;
    Z : float := Low_Z;
    Fofxyz : float;
    Pi : float := 3.141592654;

begin

  for k in 1..(Count_Z) loop
     Y := Low_Y;
     for j in 1..(Count_Y) loop
        X := Low_X;
        for i in 1..(Count_X) loop
         if (k = 1) then
           C_New(i,j,k) := 0.0;
           C_Calc(i,j,k) := 0.0;
         elsif (k = Count_Z+1) then
           C_New(i,j,k) := 0.0;
           C_Calc(i,j,k) := 0.0;
         elsif (j = 1) then
           C_New(i,j,k) := 0.0;
           C_Calc(i,j,k) := 0.0;
         elsif (j = Count_Y+1) then
           C_New(i,j,k) := 0.0;
           C_Calc(i,j,k) := 0.0;
         elsif (i = 1) then
           C_New(i,j,k) := 0.0;
           C_Calc(i,j,k) := 0.0;
         elsif (i = Count_X+1) then
           C_New(i,j,k) := 0.0;
           C_Calc(i,j,k) := 0.0;
         else
           if (T = 0.0 and X = 0.0 and Y <= 0.0
               and (Y + Del_Y) > 0.0
               and Z <= Height and (Z + Del_Z) > Height)
           then
               Fofxyz := Quantity;
           else
               Fofxyz := 0.0;
           end if;
```

```
--      Mu : float := (A*Del_T)/(2.0*Del_X);
--      Alpha : float := (K1*Del_T)/(Del_X*Del_X);
--      Beta : float := (K2*Del_T)/(Del_Y*Del_Y);
--      Gamma : float := (K3*Del_T)/(Del_Z*Del_Z);
            C_New(i,j,k) := C_N(i,j,k) - Mu*(C_N(i+1,j,k) -
                                             C_N(i-1,j,k)) +
                            Alpha*(C_N(i+1,j,k) - 2.0*C_N(i,j,k) +
                                   C_N(i-1,j,k)) +
                            Beta*(C_N(i,j+1,k) - 2.0*C_N(i,j,k) +
                                  C_N(i,j-1,k)) +
                            Gamma*(C_N(i,j,k+1) - 2.0*C_N(i,j,k) +
                                   C_N(i,j,k-1))+
                            Fofxyz*Del_T;
            if T /= 0.0  then
               C_Calc(i,j,k) := (Quantity/(8.0*sqrt(K1*K2*K3)
                                *sqrt(Pi*Pi*Pi*T*T*T)))*
                               exp(-((X-A*T)*(X-A*T)/(4.0*K1*T)) -
                                   (Y*Y/(4.0*K2*T)) -
                                   (Z*Z/(4.0*K3*T)));
            end if;
          end if;
            X := X + Del_X;
         end loop;
         Y := Y + Del_Y;
       end loop;
       Z := Z + Del_Z;
    end loop;
    for i in 1..Count_X loop
      for j in 1..Count_Y loop
         for k in 1..Count_Z loop
           C_N(i,j,k) := C_New(i,j,k);
         end loop;
      end loop;
    end loop;
end Compute_Final;
-------------------------------------------------------------------
--
-- PROCEDURE: Print_Info
-- DESCRIPTION:  This procedure prints the output of the program.
--  It prints out the matrix with the points for the time
--  iteration given.
-- INPUT PARAMETERS:  C_New : Tri_Matrix for calculating answer
```

```
--                    Del_X : incrementor for space
--                    Del_Y : incrementor for Y direction
--                    Del_Z : incrementor for Z direction
--                    Del_T : incrementor for time
--                    Low_X : lower bound for X
--                    Hi_X  : upper bound for X
--                    Low_Y : lower bound for Y
--                    Hi_Y  : upper bound for Y
--                    Low_Z : lower bound for Z
--                    Hi_Z  : upper bound for Z
-- OUTPUT PARAMETERS: C_New : Tri_Matrix answer
--                    File  : output file with values of matrix
-- LOCAL VARIABLES:  Count_X, Count_Y, Count_Z : integer counters
--                   X_Value, Y_Value, Z_Value : X,Y,and Z points
--                   K1, K2, K3 : diffusivity coefficients
--            B1Mu, B2Mu, B3Mu : Stability requirements <=1/8
--            Alpha1, Alpha2, Alpha3 : Oscilatory condition <= 1
-- GLOBALS USED:  None.
-- CALLED BY: main.
-- CALLS:  none.
-------------------------------------------------------------------
procedure Print_Info(C_New : in out Tri_Matrix;
             C_Calc  : in out Tri_Matrix;
                  Del_X, Del_Y, Del_Z, Del_T, A : in float;
                  Low_X, Hi_X, Low_Y, Hi_Y : in float;
                  Low_Z, Hi_Z, Time  : in float;
                  Quantity, Height : in float;
                  File : in out Text_Io.File_Type) is

     K1 : float := 1.0;
     K2 : float := 1.0;
     K3 : float := 1.0;
     Count_X  : integer := integer((Hi_X - Low_X)/Del_X);
     Count_Y  : integer := integer((Hi_Y - Low_Y)/Del_Y);
     Count_Z  : integer := integer((Hi_Z - Low_Z)/Del_Z);
     X_Value : float := Low_X;
     Y_Value : float := Low_Y;
     Z_Value : float := Low_Z;
     B1Mu : float := (K1*Del_T)/(Del_X*Del_X);
     B2Mu : float := (K2*Del_T)/(Del_X*Del_X);
     B3Mu : float := (K3*Del_T)/(Del_X*Del_X);
     Alpha1 : float := (A*Del_X)/(2.0*K1);
```

```
      Alpha2 : float := (A*Del_X)/(2.0*K2);
      Alpha3 : float := (A*Del_X)/(2.0*K3);

begin
   Text_Io.put_line("#The following are the results using the
forward time ");
   Text_Io.put_line("# central space for the partial diferential
equation:");
   Text_Io.put_line("#    Ct + A*Cx = K1*Cxx + K2*Cyy + K3*Czz +
F(X,Y,Z)");
   Text_Io.put_line("# The resulting matrix is: ");
   Text_Io.new_line;
   Text_Io.new_line(File,2);
   Text_Io.put(File,"#  Output for threedprob.a ");
   Text_Io.new_line(File);
   Text_Io.put_line(File,"#  Ct + A*Cx = K1*Cxx + K2*Cyy + K3*Czz
+ F(X,Y,Z)");
   Text_Io.new_line(File);
   Text_Io.put(File,"#  Min_X = ");
   My_Float_Io.put(File,Low_X,3,2,0);
   Text_Io.put(File,",  Max_X = ");
   My_Float_Io.put(File,Hi_X,3,2,0);
   Text_Io.put(File,",  Delta_X = ");
   My_Float_Io.put(File,Del_X,3,5,0);
   Text_Io.new_line(File);
   Text_Io.put(File,"#  Min_Y = ");
   My_Float_Io.put(File,Low_Y,3,2,0);
   Text_Io.put(File,",  Max_Y = ");
   My_Float_Io.put(File,Hi_Y,3,2,0);
   Text_Io.put(File,",  Delta_Y = ");
   My_Float_Io.put(File,Del_Y,3,5,0);
   Text_Io.new_line(File);
   Text_Io.put(File,"#  Min_Z = ");
   My_Float_Io.put(File,Low_Z,3,2,0);
   Text_Io.put(File,",  Max_Z = ");
   My_Float_Io.put(File,Hi_Z,3,2,0);
   Text_Io.put(File,",  Delta_Z = ");
   My_Float_Io.put(File,Del_Z,3,5,0);
   Text_Io.new_line(File);
   Text_Io.put(File,"#  Delta_T = ");
   My_Float_Io.put(File,Del_T,3,3,0);
   Text_Io.new_line(File);
```

```
Text_Io.put(File,"#  Time of calculation = ");
My_Float_Io.put(File,Time,3,3,0);
Text_Io.new_line(File);
Text_Io.put(File,"#  A = ");
My_Float_Io.put(File,A,3,2,0);
Text_Io.new_line(File);
Text_Io.put(File,"#  Emission Value = ");
My_Float_Io.put(File,Quantity,3,2,0);
Text_Io.new_line(File);
Text_Io.put(File,"#  Stack height = ");
My_Float_Io.put(File,Height,3,2,0);
Text_Io.new_line(File);
Text_Io.put(File,"#  B1Mu = (K1*Del_T)/(Del_X*Del_X) = ");
My_Float_Io.put(File,B1Mu,3,4,0);
Text_Io.new_line(File);
Text_Io.put(File,"#  B2Mu = (K2*Del_T)/(Del_X*Del_X) = ");
My_Float_Io.put(File,B2Mu,3,4,0);
Text_Io.new_line(File);
Text_Io.put(File,"#  B3Mu = (K3*Del_T)/(Del_X*Del_X) = ");
My_Float_Io.put(File,B3Mu,3,4,0);
Text_Io.new_line(File);
Text_Io.put(File,"#  Alpha1 = (A*Del_X)/(2*K1) = ");
My_Float_Io.put(File,Alpha1,3,4,0);
Text_Io.new_line(File);
Text_Io.put(File,"#  Alpha2 = (A*Del_X)/(2*K2) = ");
My_Float_Io.put(File,Alpha2,3,4,0);
Text_Io.new_line(File);
Text_Io.put(File,"#  Alpha3 = (A*Del_X)/(2*K3) = ");
My_Float_Io.put(File,Alpha3,3,4,0);
Text_Io.new_line(File);
Text_Io.set_col(File,1);
Text_Io.put(File,"# X ");
Text_Io.set_col(File,10);
Text_Io.put(File," Y ");
Text_Io.set_col(File,20);
Text_Io.put(File," Z ");
Text_Io.set_col(File,30);
Text_Io.put(File,"Numerical");
Text_Io.set_col(File,47);
Text_Io.put(File,"Actual");
Text_Io.new_line(File);
for i in 1..Count_X+1 loop
```

```
      Z_Value := Low_Z;
      for k in 1..Count_Z+1 loop
         Y_Value := Low_Y;
         for j in 1..Count_Y+1 loop
           Text_Io.set_col(File,1);
              My_Float_Io.put(File,X_Value,1,2,0);
           Text_Io.set_col(File,10);
              My_Float_Io.put(File, Y_Value,1,2,0);
           Text_Io.set_col(File,20);
              My_Float_Io.put(File, Z_Value,1,2,0);
           Text_Io.set_col(File,30);
              My_Float_Io.put(File,C_New(i,j,k),1,6,2);
              Text_Io.set_col(File,47);
              My_Float_Io.put(File,C_Calc(i,j,k),1,6,2);
              Text_Io.new_line(File);
              Y_Value := Y_Value + Del_Y;
         end loop;
         Z_Value := Z_Value + Del_Z;
      end loop;
      X_Value := X_Value + Del_X;
   end loop;
     Text_Io.new_line;
  end Print_Info;
-------------------------------------------------------------------
-- PROCEDURE:  Main
-- DESCRIPTION: This is the main program and calls all the above
--              modules
-- INPUT PARAMETERS: none
-- OUTPUT PARAMETERS: none
-- LOCAL VARIABLES: none
-- GLOBALS USED: All.
-- CALLED BY:  none
-- CALLS:  Get_Info, Compute_Final, Print_Info.
-------------------------------------------------------------------
begin --MAIN
   Text_Io.Create(Outfile,Text_Io.Out_File,"test3d.dat");
   Get_Info (Delta_X, Delta_Y, Delta_Z, Delta_T, Min_X, Max_X,
             Min_Y, Max_Y, Min_Z, Max_Z, A, T_Value,
             Q_Value, Stk_Hgt);
   T := 0.0;
   while T <= T_Value loop
     Compute_Final(C_N, C_New, C_Calc, T, Delta_X, Delta_Y,
```

```
                Delta_Z, Delta_T, Min_X, Max_X, Min_Y, Max_Y,
                Min_Z, Max_Z, A, Q_Value, Stk_Hgt);
      T := T + Delta_T;
    end loop;
    Print_Info(C_New, C_Calc, Delta_X, Delta_Y, Delta_Z, Delta_T,
               A, Min_X, Max_X,Min_Y,Max_Y,Min_Z,Max_Z,T_Value,
               Q_Value,Stk_Hgt,Outfile);
    Text_Io.Close(Outfile);
end threed;
```

B.5 Steady-State Equation Ada Code

```
-- FILE: ubareqkcon.a
-- PROJECT:  Thesis work, example calculation
-- DATE: 15 Sep 93
-- VERSION: Version 1.0
-- AUTHORS:  Capt Dave Paal
-- DESCRIPTION:  This program solves a partial differential
--     equation using the forward time and central space.
--     The partial differential equation to be solved is:
--
--          U_Bar*Cx = KY*Cyy + KZ*Czz
--
--     where KY/U_Bar = KZ/U_Bar = 1.0
--     and (Ki*Del_T)/(U_Bar*Del_X*Del_X)<= 1/4
--
--  with C(0,x,y) = sin((Pi/20)*(Y + 10)) * sin((Pi/20)*(Z + 10))
--     and  C(x,y,z) = 0.0 at Y=+-10 and Z=+-10
-- OPERATING SYSTEM:  UNIX/Sun Sparc Station
-- LANGUAGE:  Meridian Ada
-- FILES USED: Output:  kconequ.dat.
-------------------------------------------------------------------
--
-- CONTEXT CLAUSES
--
with text_io;
with Math_Lib;
use Math_Lib;
with my_integer_io;
with my_float_io;

procedure kconequbar  is
--
-- TYPE DECLARATION
--
   type Matrix is array (0..90,0..90) of float;
--
--
-- GLOBAL VARIABLES AND EXCEPTIONS
   C_N          : Matrix;
   C_New        : Matrix;
   C_Calc       : Matrix;
   X            : integer;
```

```
   Delta_X      : float;
   Delta_Y      : float;
   Delta_Z      : float;
   Min_X        : float;
   Max_X        : float;
   Min_Y        : float;
   Max_Y        : float;
   Min_Z        : float;
   Max_Z        : float;
   U_Bar        : float;
   Outfile      : Text_Io.File_Type;
-------------------------------------------------------------------
-- PROCEDURE:  Get information for calculation
-- DESCRIPTION:  This procedure gets the values of Del_X, Del_Y,
--   Del_Z, Low_X, Hi_X, Low_Y, Hi_Y, Low_Z, Hi_Z, U_Bar,
--   Quantity, Height.
-- INPUT PARAMETERS: None.
-- OUTPUT PARAMETERS:  Del_X  : incrementor for space
--                     Del_Y  : incrementor for Y direction
--                     Del_Z  : incrementor for Z direction
--                     Low_X  : lower bound for X
--                     Hi_X   : upper bound for X
--                     Low_Y  : lower bound for Y
--                     Hi_Y   : upper bound for Y
--                     Low_Z  : lower bound for Z
--                     Hi_Z   : upper bound for Z
--                     Ubar   : constant wind speed
-- LOCAL VARIABLES:  None.
-- GLOBALS USED: None.
-- CALLED BY: main.
-- CALLS:   none.
-------------------------------------------------------------------
procedure Get_Info (Del_X : in out float;
    Del_Y : in out float;
    Del_Z : in out float;
                    Low_X : in out float;
                    Hi_X  : in out float;
                    Low_Y : in out float;
                    Hi_Y  : in out float;
                    Low_Z : in out float;
                    Hi_Z  : in out float;
                    Ubar  : in out float) is
```

```
   begin

     Text_Io.put_line("Enter the value (Floating point) of Delta
X. ");
     My_Float_Io.get(Del_X);
     Text_Io.put_line("Enter the value (Floating point) of Delta
Y. ");
     My_Float_Io.get(Del_Y);
     Text_Io.put_line("Enter the value (Floating point) of Delta
Z. ");
     My_Float_Io.get(Del_Z);
     Text_Io.put_line("Enter the minimum value (Floating point)
of X. ");
     My_Float_Io.get(Low_X);
     Text_Io.put_line("Enter the value (Floating point) of X
wanted. ");
     My_Float_Io.get(Hi_X);
     Text_Io.put_line("Enter the minimum value (Floating point)
of Y. ");
     My_Float_Io.get(Low_Y);
     Text_Io.put_line("Enter the maximum value (Floating point)
of Y. ");
     My_Float_Io.get(Hi_Y);
     Text_Io.put_line("Enter the minimum value (Floating point)
of Z. ");
     My_Float_Io.get(Low_Z);
     Text_Io.put_line("Enter the maximum value (Floating point)
of Z. ");
     My_Float_Io.get(Hi_Z);
     Text_Io.put_line("Enter the value (Floating point) of U_Bar.
");
     My_Float_Io.get(Ubar);
  end Get_Info;
-----------------------------------------------------------------
--
-- PROCEDURE: Compute final Matrix
-- DESCRIPTION:  This procedure calculates final answer through
--  iterations using a finite difference scheme with forward
--  time/central space.
-- INPUT PARAMETERS:  C_N : Matrix for C_N(i,j)
--                    C_New : Matrix for calculating numerical
```

```
--                              answer
--                      C_Calc : Matrix for calculating exact
--                               answer
--                      Del_X : incrementor for space
--                      Del_Y : incrementor for Y direction
--                      Del_Z : incrementor for Z direction
--                      Low_X : lower bound for X
--                      Hi_X  : upper bound for X
--                               where calculation is made
--                      Low_Y : lower bound for Y
--                      Hi_Y  : upper bound for Y
--                      Low_Z : lower bound for Z
--                      Hi_Z  : upper bound for Z
--                      Ubar  : constant wind speed
-- OUTPUT PARAMETERS: C_N   : Matrix for C(N)
--                      C_New : Matrix for calculating numerical
--                              answer
--                      C_Calc : Matrix for calculating exact
--                               answer
-- LOCAL VARIABLES: KY      : Ubar*sigmaY^2/2*X
--                  KZ      : Ubar*sigmaZ^2/2*X
--                  Alpha   : (KY*Del_X)/(Ubar*Del_Y*Del_Y)
--                  Beta    : (KZ*Del_X)/(Ubar*Del_Z*Del_Z)
--                  Count_Y : integer((Hi_Y -Low_Y)/Del_Y)
--                  Count_X : integer((Hi_X - Low_X)/Del_X)
--                  Count_Z : integer((Hi_Z - Low_Z)/Del_Z)
--                  X       : value of x
--                  Y       : value of y
--                  Z       : value of z
--                  Pi      : the number Pi
-- GLOBALS USED: None.
-- CALLED BY:  main.
-- CALLS:   none.
-------------------------------------------------------------------
procedure Compute_Final(C_N : in out Matrix;
   C_New : in out Matrix;
   C_Calc : in out Matrix;
                  X : in integer;
                  Del_X, Del_Y, Del_Z : in float;
                  Low_X, Hi_X, Low_Y, Hi_Y : in float;
                  Low_Z, Hi_Z : in float;
                  Ubar  : in float) is
```

```
    KY : float := Ubar;
    KZ : float := Ubar;
    Count_X  : integer := integer((Hi_X - Low_X)/Del_X);
    Count_Y  : integer := integer((Hi_Y - Low_Y)/Del_Y);
    Count_Z  : integer := integer((Hi_Z - Low_Z)/Del_Z);
    X_Value : float := Low_X + float(X-1)*Del_X;
    Y : float := Low_Y;
    Z : float := Low_Z;
    Alpha : float := (KY*Del_X)/(Ubar*Del_Y*Del_Y);
    Beta : float := (KZ*Del_X)/(Ubar*Del_Z*Del_Z);
    Fofxyz : float;
    Pi : float := 3.141592654;
    PiDiv20 : float := Pi/20.0;

begin
  Z := Low_Z;
  for k in 1..(Count_Z) loop
     Y := Low_Y;
     for j in 1..(Count_Y) loop
       if (k = 1) then
          C_New(j,k) := 0.0;
       elsif (k = Count_Z+1) then
          C_New(j,k) := 0.0;
       elsif (j = 1) then
          C_New(j,k) := 0.0;
       elsif (j = Count_Y+1) then
          C_New(j,k) := 0.0;
       elsif (X_Value = 0.0)  then
          C_New(j,k) :=
   sin((PiDiv20)*(Y+10.0))*sin((PiDiv20)*(Z+10.0));
       else
--     Alpha : float := (KY*Del_X)/(Ubar*Del_Y*Del_Y);
--     Beta : float := (KZ*Del_X)/(Ubar*Del_Z*Del_Z);
          C_New(j,k) := C_N(j,k) +
                        Alpha*(C_N(j+1,k) - 2.0*C_N(j,k) +
                               C_N(j-1,k)) +
                        Beta*(C_N(j,k+1) - 2.0*C_N(j,k) +
                              C_N(j,k-1));
       end if;
       C_Calc(j,k) := exp((-(Pi*Pi)*X_Value)/20))*
            sin((PiDiv20)*(Y+10.0))*sin((PiDiv20)*(Z+10.0));
```

```
        Y := Y + Del_Y;
      end loop;
      Z := Z + Del_Z;
    end loop;
    for k in 1..(Count_Z) loop
      for j in 1..(Count_Y) loop
        C_N(j,k) := C_New(j,k);
      end loop;
    end loop;
end Compute_Final;
-------------------------------------------------------------------
--
-- PROCEDURE: Print_Info
-- DESCRIPTION:  This procedure prints the output of the program.
--  It prints out the matrix with the points for the time
--  iteration given.
-- INPUT PARAMETERS:  C_New : Matrix for calculating numerical
--                            answer
--                    C_Calc : Matrix for calculating exact
--                             answer
--                    Del_X : incrementor for space
--                    Del_Y : incrementor for Y direction
--                    Del_Z : incrementor for Z direction
--                    U_Bar : advection constant
--                    Low_X : lower bound for X
--                    Hi_X  : upper bound for X
--                    Low_Y : lower bound for Y
--                    Hi_Y  : upper bound for Y
--                    Low_Z : lower bound for Z
--                    Hi_Z  : upper bound for Z
-- OUTPUT PARAMETERS: C_New : Matrix numerical answer
--                    C_Calc : Matrix for calculating exact
--                             answer
--                    File  : output file has values of matrix
-- LOCAL VARIABLES:  Count_X, Count_Y, Count_Z : integer counters
--                  X_Value, Y_Value, Z_Value : X,Y, and Z points
-- GLOBALS USED:  None.
-- CALLED BY: main.
-- CALLS:  none.
-------------------------------------------------------------------
procedure Print_Info(C_New : in out Matrix;
                     C_Calc : in out Matrix;
```

```
   Del_X, Del_Y, Del_Z, Ubar : in float;
                  Low_X,Hi_X,Low_Y,Hi_Y,Low_Z,Hi_Z : in float;
                  File : in out Text_Io.File_Type) is
      KY : float := Ubar;
      KZ : float := Ubar;
      Count_X  : integer := integer((Hi_X - Low_X)/Del_X);
      Count_Y  : integer := integer((Hi_Y - Low_Y)/Del_Y);
      Count_Z  : integer := integer((Hi_Z - Low_Z)/Del_Z);
      X_Value : float := Low_X;
      Y_Value : float := Low_Y;
      Z_Value : float := Low_Z;
      StableY : float := (KY*Del_X)/(Ubar*Del_Y*Del_Y);
      StableZ : float := (KZ*Del_X)/(Ubar*Del_Z*Del_Z);

begin
   Text_Io.put_line("#The following are the results using the
forward time ");
   Text_Io.put_line("# central space for the partial diferential
equation:");
   Text_Io.put_line("#        Ubar*Cx = KY*Cyy + KZ*Czz ");
   Text_Io.put_line("# The resulting matrix is: ");
   Text_Io.new_line;
   Text_Io.new_line(File,2);
   Text_Io.put(File,"#  Output from file ubareqkcon.a ");
   Text_Io.new_line(File);
   Text_Io.put_line(File,"#  Ubar*Cx = KY*Cyy + KZ*Czz ");
   Text_Io.new_line(File);
   Text_Io.put_line(File,"#  KY and KZ are constant and equal
U_bar");
   Text_Io.new_line(File);
   Text_Io.put(File,"#  Min_X = ");
   My_Float_Io.put(File,Low_X,3,2,0);
   Text_Io.put(File,",  Max_X = ");
   My_Float_Io.put(File,Hi_X,3,2,0);
   Text_Io.put(File,",  Delta_X = ");
   My_Float_Io.put(File,Del_X,3,5,0);
   Text_Io.new_line(File);
   Text_Io.put(File,"#  Min_Y = ");
   My_Float_Io.put(File,Low_Y,3,2,0);
   Text_Io.put(File,",  Max_Y = ");
   My_Float_Io.put(File,Hi_Y,3,2,0);
   Text_Io.put(File,",  Delta_Y = ");
```

```
My_Float_Io.put(File,Del_Y,3,5,0);
Text_Io.new_line(File);
Text_Io.put(File,"#  Min_Z = ");
My_Float_Io.put(File,Low_Z,3,2,0);
Text_Io.put(File,",  Max_Z = ");
My_Float_Io.put(File,Hi_Z,3,2,0);
Text_Io.put(File,",  Delta_Z = ");
My_Float_Io.put(File,Del_Z,3,5,0);
Text_Io.new_line(File);
Text_Io.put(File,"#  U_Bar = ");
My_Float_Io.put(File,Ubar,3,5,0);
Text_Io.new_line(File);
Text_Io.put(File,"#  Y stability value = ");
My_Float_Io.put(File,StableY,3,5,0);
Text_Io.new_line(File);
Text_Io.put(File,"#  Z stability value = ");
My_Float_Io.put(File,StableZ,3,5,0);
Text_Io.new_line(File);
Text_Io.set_col(File,1);
Text_Io.put(File,"# X ");
Text_Io.set_col(File,10);
Text_Io.put(File," Y ");
Text_Io.set_col(File,20);
Text_Io.put(File," Z ");
Text_Io.set_col(File,30);
Text_Io.put(File,"Numerical");
Text_Io.set_col(File,47);
Text_Io.put(File,"Actual");
Text_Io.new_line(File);
   for k in 1..Count_Z+1 loop
      Y_Value := Low_Y;
      for j in 1..Count_Y+1 loop
   Text_Io.set_col(File,1);
           My_Float_Io.put(File,Hi_X,1,2,0);
   Text_Io.set_col(File,10);
           My_Float_Io.put(File, Y_Value,1,2,0);
   Text_Io.set_col(File,20);
           My_Float_Io.put(File, Z_Value,1,2,0);
           Text_Io.set_col(File,30);
           My_Float_Io.put(File,C_New(j,k),1,6,2);
           Text_Io.set_col(File,47);
           My_Float_Io.put(File,C_Calc(j,k),1,6,2);
```

```
            Text_Io.new_line(File);
            Y_Value := Y_Value + Del_Y;
         end loop;
         Z_Value := Z_Value + Del_Z;
      end loop;
     Text_Io.new_line;
  end Print_Info;
-------------------------------------------------------------------
-- PROCEDURE:  Example1.3.1 (a.k.a main)
-- DESCRIPTION: This is the main program and calls all the above
--           modules
-- INPUT PARAMETERS: none
-- OUTPUT PARAMETERS: none
-- LOCAL VARIABLES: none
-- GLOBALS USED: All.
-- CALLED BY:  none
-- CALLS:  Get_Info, Compute_Final, Print_Info.
-------------------------------------------------------------------
begin --MAIN
   Text_Io.Create(Outfile,Text_Io.Out_File,"kconequ.dat");
   Get_Info (Delta_X, Delta_Y, Delta_Z, Min_X, Max_X, Min_Y,
         Max_Y, Min_Z, Max_Z, U_Bar);
   X := integer((Max_X - Min_X)/Delta_X)+1;
   for I in 1..X loop
     Compute_Final(C_N, C_New, C_Calc, I, Delta_X, Delta_Y,
       Delta_Z, Min_X, Max_X, Min_Y, Max_Y, Min_Z, Max_Z, U_Bar);
   end loop;
   Print_Info(C_New, C_Calc, Delta_X, Delta_Y, Delta_Z, U_Bar,
             Min_X, Max_X, Min_Y, Max_Y, Min_Z, Max_Z, Outfile);
   Text_Io.Close(Outfile);
end kconequbar;
```

Appendix C. 1-D diffusion and 3-D steady state equations

This appendix describes the one-dimensional diffusion equation and the three-dimensional steady state equations and how they used to this research.

C.1 The Diffusion Equation:

The one-dimensional diffusion equation (C.1), also known as the one-dimensional heat equation, is the simplest parabolic partial differential equation.

$$c_t = bc_{xx} \tag{C.1}$$

The finite difference scheme used in this research to solve equation (C.1) is the forward-time central-space scheme

$$\frac{c_m^{n+1} - c_m^n}{\Delta t} = k_x \frac{c_{m+1}^n - 2c_m^n + c_{m-1}^n}{\Delta x^2} \tag{C.2}$$

which can also be written as

$$c_m^{n+1} = c_m^n + k_x \mu (c_{m+1}^n - 2c_m^n + c_{m-1}^n) \tag{C.3}$$

where $\mu = \Delta t / \Delta x^2$. The stability condition for this method is $k_x \mu \leq 1/2$. It is found by replacing c_m^n with $g^n e^{im\theta}$ in Equation C.2 leaving

$$\frac{g-1}{\Delta t} = k_x \frac{e^{i\theta} - 2 + e^{-i\theta}}{\Delta x^2} \tag{C.4}$$

and solving for g gives

$$g = 1 - 4k_x\mu \sin^2 \frac{1}{2}\theta. \tag{C.5}$$

The condition $|g| \leq 1$ from Theorem 2.2.1 in Strikwerda (26:42) is equivalent to $4k_x\mu \sin^2 \frac{1}{2}\theta \leq 2$ which is true for all θ if and only if $k_x\mu \leq 1/2$. Thus this method is conditionally stable.

C.2 3-D Steady State Equation

The Eulerian approach (25:542) to the three- dimensional steady state equation examined in this research is

$$\bar{u}c_x = k_x c_{xx} + k_y c_{yy} + k_z c_{yy} + q \tag{C.6}$$

with

$$c(x, y, z) = 0 \quad x, y, z \rightarrow \pm\infty \tag{C.7}$$

where $\bar{u}$ is the wind speed (positive constant), $k_x = k_y = k_z$ are constant in this research and are the x-axis, y-axis, and z-axis diffusivity terms respectively, and q is

the source term. The forward-time central-space finite difference scheme (C.8) used to solve equation (C.6) is

$$\frac{c_{i,j,k}^{n+1} - c_{i,j,k}^{n}}{\Delta t} + \bar{u}\frac{c_{i+1,j,k}^{n} - c_{i-1,j,k}^{n}}{2\Delta x} = k_x \frac{c_{i+1,j,k}^{n} - 2c_{i,j,k}^{n} + c_{i-1,j,k}^{n}}{\Delta x^2} + k_y \frac{c_{i,j+1,k}^{n} - 2c_{i,j,k}^{n} + c_{i,j-1,k}^{n}}{\Delta y^2} + k_z \frac{c_{i,j,k+1}^{n} - 2c_{i,j,k}^{n} + c_{i,j,k-1}^{n}}{\Delta z^2}. \quad \text{(C.8)}$$

The solution to eq. (C.8) would be compared to the exact solution (25:542) of

$$c = \frac{q}{4\pi k_i r} \exp[-\frac{\bar{u}(r-x)}{2k_i}] \quad \text{(C.9)}$$

where q = the source term, $k_i = (k_x k_y k_z)^{1/3}$, and $r = (x^2 + y^2 + z^2)^{1/2}$

The stability condition is the similar to the three-dimensional advection-diffusion discussed in Section 3.4.4. It is assumed that $k_x c_{xx} \ll \bar{u} c_x$, that is, the diffusion in the x-axis direction is much less than the advection in the x- axis direction and is therefore negligible. It is for this reason that this equation was not looked at in detail.

Bibliography

1. Benarie, Michael M. "The limits of air pollution modeling," *Atmospheric Environment*, *21*(1):1-5 (1987).
2. Bierly, E. W. and E. W. Hewson. "Some restrictive meteorological conditions to be considered in the design of stacks," *Journal of Applied Meteorology*, *1*(3):383-390 (1962).
3. Burden, Richard L. and J. Douglas Faires. *Numerical Analysis*. Boston, Massachusetts: PWS-KENT publishing company, 1989.
4. Chock, David P. "A comparison of numerical methods for solving the advection equation-II," *Atmospheric Environment*, *19*(4):571-586 (1985).
5. Chock, David P. "A comparison of numerical methods for solving the advection equation-III," *Atmospheric Environment*, *25A*(5/6):853-871 (1991).
6. Chock, David P. and A. M. Dunker. "A comparison of numerical methods for solving the advection equation," *Atmospheric Environment*, *17*(1):11-24 (1983).
7. Dop, Han Van. "Buoyant Plume Rise in a Lagrangian Framework," *Atmospheric Environment*, *26A*(7):1335-1346 (1992).
8. Draxler, R.R., R. Dietz, et al. "Across North America Tracer Experiment (ANTEX): Sampling and Analysis," *Atmospheric Environment*, *25A*(12):2815-2836 (1991).
9. Egana, B.A. and J.R.Mahoney. "Numerical modeling of advection and diffusion of urban area source pollutants," *Journal of Applied Meteorology*, *11*:312-322 (January 1972).
10. EPA. *SCREEN Model User's Guide*. Technical Report, Research Triangle Park, NC 27711: U. S. Environmental Protection Agency, April 1992. EPA-450/4-92-006.
11. EPA. *Toxic Modeling System Long-Term (TOXLT) User's Guide*. United States Environmental Protection Agency, Research Trianle Park, NC 27711, 1992. EPA-450/4-92-003.
12. EPA. *Toxic Modeling System Short-Term (TOXST) User's Guide*. United States Environmental Protection Agency, Research Trianle Park, NC 27711, 1992. EPA-450/4-92-002.
13. Eppel, D.P. and others. "A Numerical Model For Simulating Pollutant Transport From a Single Point Source," *Atmospheric Environment*, *25A*(7):1391-1401 (1991).
14. Henderson-Sellers, B. and S.E. Allen. "Verification of the plume rise/dispersion model," *Ecological Modeling*, *30*:209-277 (1985).

15. Hoult, D.P., et al. "A theory of plume rise compared with field observatoins," *Journal of Air Pollution Control Association*, *19*:585–590 (1969).

16. Kasibhatla, Prasad S., Peters L. K. and Fairweather G. "Numerical simulation of transprot from an infinite line source: error analysis," *Atmospheric Environment*, *22*(1):75–82 (1988).

17. Kasibhatla, Prasad S. and Leonard K. Peters. "Numerical Simulation of Transport from a Point Source: Error Analysis," *Atmospheric Environment*, *24A*(3):693–702 (1990).

18. Krishnamurthy, Ramesh and J. Gordon Hall. "Numerical and approximate analytical solutions for plume rise," *Atmospheric Environment*, *21*(10):2083–2089 (1987).

19. Kunkel, B. A. *AFTOX - The Air Force Toxic Chemical Dispersion Model - A User's Guide*, 1991. PL-TR-91-2119, ADA246726.

20. Lamb, R.G. "Air pollution models as descriptors of cause-effect relationships," *Atmospheric Environment*, *18*(3):591–606 (1984).

21. McRae, Gregory J., William R. Goodin and John H. Seinfeld. "Numerical solution of the atmospheric diffusion equation for chemically reacting flows," *Journal of Computational Physics*, *45*(1):1–42 (1982).

22. Okamoto, Shin'ichi and Kiyoshige Shiozawa. "Trajectory plume model for simulation of air pollution transients," *Atmospheric Environment*, *21*(10):2145–2152 (1987).

23. Reynolds, Steven D., Philip M. Roth and John H. Seinfeld. "Mathematical modeling of photochemical air pollution," *Atmospheric Environment*, *7*(7):1033–1061 (1973).

24. Schohl, G.A. and F.M. Holly Jr. "Cubic-spline interpolation in Lagrangian advection computation," *Journal of Hydraulic Engineering*, *117*:248–253 (February 1991).

25. Seinfeld, John H. *Atmospheric Chemistry and Physics*. New York, NY: John Wiley and Sons, 1986.

26. Strikwerda, John C. *Finite Difference Schemes and Partial Differential Equations*. Pacific Grove, CA 93950: Wadsworth and Brooks, 1989.

27. Turner, D. Bruce. "Workbook of Atmospheric Dispersion Estimates,". U. S. Department of Health, Education and Welfare, Cincinnati, OH, 1969. Public Health Service Publication No. 999-AP-26.

28. Young, David M. and Robert T. Gregory. *A survey of numerical mathematics*. Reading, MS: Addison-Wesley Publishing Company, 1988.

29. Zannetti, Paolo. *Air pollution modeling: theories, computational methods and available software*. New York, NY: Van Nostrand Reinhold, 1990.

Vita

Capt Paal was born in San Antonio, Texas on 9 December 1962. He attended the College of Saint Thomas in Saint Paul, Minnesota and graduated with a Bachelor of Arts in Mathematics in 1987. Capt Paal attended the Air Force Institute of Technology (AFIT) at Texas A&M University for the Basic Meteorology Program in 1988. In 1989 Capt Paal was assigned to the Air Force Global Weather Central, Offutt Air Force Base, Omaha, Nebraska as Production Duty Officer. In 1992 Capt Paal was selected to attend AFIT in residence at Wright Patterson Air Force Base, Dayton, Ohio for completion of a Master of Science in Computer Systems.

Permanent address: 312 Hickory Street
Chaska, Minnesota 55318

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1993	Master's Thesis

4. TITLE AND SUBTITLE
AIR POLLUTION TRANSPORT MODELING

5. FUNDING NUMBERS

6. AUTHOR(S)
David M. Paal, Captain, USAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology, WPAFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/ENC/GCS/93D-1

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This research effort addresses modeling of the transportation of air pollution in the atmosphere and the numerical analysis of the partial differential equations used in such modeling. Three Gaussian models are examined and compared using example problems. Several finite difference schemes are developed to solve the partial differential equations used in air pollution transport modeling. This study examines three Gaussian models: SCREEN, AFTOX, and the program GAUSPLUM. The model GAUSPLUM is developed in this study and uses the Ada programming language and the analytic solution to the advection-diffusion equation. Numerical analysis of the partial differential equations (PDE) used in air pollution modeling is also examined. The equations are generally parabolic or hyperbolic PDE's. The following are examined in this research: the advection equation; the one-, two-, and three-dimensional advection-diffusion equations; and the two-dimensional steady-state equation.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Air Pollution Transport, Modeling, Finite Difference Scheme, Stability, Consistency, Convergence, Advection-Diffusion Equations	119
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL